Restricted Observational Equivalence

Petar Paradžik Ante Derek

University of Zagreb
Faculty of Electrical Engineering and Computing

LAP 2021 @Dubrovnik,Croatia
September 20-24

Motivation: off-line guessing resistance

Example

The following protocol aims at to authenticate Alice (a) to Bob (b) using a password $p_{a,b}$

$$a \rightarrow b : (a, b)$$
$$b \rightarrow a : n_b$$
$$a \rightarrow b : enc(n_b, p_{a,b})$$

Motivation: off-line guessing resistance

Example

The following protocol aims at to authenticate Alice (a) to Bob (b) using a password $p_{a,b}$

$$a \to b : (a, b)$$
$$b \to a : n_b$$
$$a \to b : enc(n_b, p_{a,b})$$

Can an attacker verify his password guess in some way?

Motivation: off-line guessing resistance

Example

The following protocol aims at to authenticate Alice (a) to Bob (b) using a password $p_{a,b}$

$$\begin{aligned} a &\rightarrow b : (a, b) \\ b &\rightarrow a : n_b \\ a &\rightarrow b : enc(n_b, p_{a,b}) \end{aligned}$$

Can an attacker verify his password guess in some way?

1. *guess:* $p'_{a,b} = 123ccas\#$

Motivation: off-line guessing resistance

Example

The following protocol aims at to authenticate Alice (a) to Bob (b) using a password $p_{a,b}$

$$a \rightarrow b : (a, b)$$
$$b \rightarrow a : n_b$$
$$a \rightarrow b : enc(n_b, p_{a,b})$$

Can an attacker verify his password guess in some way?

1. *guess:* $p'_{a,b} = 123ccas\#$
2. *decrypt:* $n'_b = dec(enc(n_b, p_{a,b}), p'_{a,b})$

Motivation: off-line guessing resistance

Example

The following protocol aims at to authenticate Alice (a) to Bob (b) using a password $p_{a,b}$

$$a \rightarrow b : (a, b)$$
$$b \rightarrow a : n_b$$
$$a \rightarrow b : enc(n_b, p_{a,b})$$

Can an attacker verify his password guess in some way?

1. *guess:* $p'_{a,b} = 123ccas\#$
2. *decrypt:* $n'_b = dec(enc(n_b, p_{a,b}), p'_{a,b})$
3. *compare:* $n'_b \stackrel{?}{=} n_b$

Motivation: off-line guessing resistance

Example

The following protocol aims at to authenticate Alice (a) to Bob (b) using a password $p_{a,b}$

$$
\begin{aligned}
a &\to b : (a, b) \\
b &\to a : n_b \\
a &\to b : enc(n_b, p_{a,b})
\end{aligned}
$$

Can an attacker verify his password guess in some way?

1. *guess:* $p'_{a,b} = 123ccas\#$
2. *decrypt:* $n'_b = dec(enc(n_b, p_{a,b}), p'_{a,b})$
3. *compare:* $n'_b \stackrel{?}{=} n_b$

An attacker can verify his guess so the protocol is not resistant to off-line gusssing!

Motivation: off-line guessing resistance

Example

Consider the following protocol.

$a \rightarrow b : (a, b)$

OK, I (b) will send you my RSA public key (e, n).

$b \rightarrow a : enc((e, n), p_{a,b})$

Motivation: off-line guessing resistance

Example

Consider the following protocol.

$$a \rightarrow b : (a, b)$$

OK, I (b) will send you my RSA public key (e, n).

$$b \rightarrow a : enc((e, n), p_{a,b})$$

Can an attacker verify his password guess in some way?

Motivation: off-line guessing resistance

Example

Consider the following protocol.

$$a \to b : (a, b)$$

OK, I (b) will send you my RSA public key (e, n).

$$b \to a : enc((e, n), p_{a,b})$$

Can an attacker verify his password guess in some way?

1. *guess:* $p'_{a,b} = 123cass\#$

Motivation: off-line guessing resistance

Example

Consider the following protocol.

$a \to b : (a, b)$

OK, I (b) will send you my RSA public key (e, n).

$b \to a : enc((e, n), p_{a,b})$

Can an attacker verify his password guess in some way?

1. *guess:* $p'_{a,b} = 123cass\#$
2. *decrypt:* $(e', n') = dec(enc((e, n), p_{a,b}), p'_{a,b})$

Motivation: off-line guessing resistance

Example

Consider the following protocol.

$$a \rightarrow b : (a, b)$$

OK, I (b) will send you my RSA public key (e, n).

$$b \rightarrow a : enc((e, n), p_{a,b})$$

Can an attacker verify his password guess in some way?

1. *guess:* $p'_{a,b} = 123cass\#$
2. *decrypt:* $(e', n') = dec(enc((e, n), p_{a,b}), p'_{a,b})$
3. *if* e' *is odd, and* n' *has no small prime factors, the guess* $p'_{a,b}$ *is (with a high probability) correct!*

Motivation: off-line guessing resistance

Example

Consider the following protocol.

$a \to b : (a, b)$

OK, I (b) will send you my RSA public key (e, n).

$b \to a : enc((e, n), p_{a,b})$

Can an attacker verify his password guess in some way?

1. *guess:* $p'_{a,b} = 123cass\#$
2. *decrypt:* $(e', n') = dec(enc((e, n), p_{a,b}), p'_{a,b})$
3. *if e' is odd, and n' has no small prime factors, the guess $p'_{a,b}$ is (with a high probability) correct!*

The protocol is not resistant to off-line gusssing!

Motivation: off-line guessing resistance

Example

Consider the PK-EKE protocol where pk is an asymmetric key and k is a symmetric session key.

$$
\begin{aligned}
a \rightarrow b &: (a, b, enc(pk, p_{a,b})) \\
b \rightarrow a &: enc(enc(k, pk), p_{a,b}) \\
a \rightarrow b &: enc(n_a, k) \\
b \rightarrow a &: enc((n_a, n_b), k) \\
a \rightarrow b &: enc(n_b, k)
\end{aligned}
$$

Motivation: off-line guessing resistance

Example

Consider the PK-EKE protocol where pk *is an asymmetric key and* k *is a symmetric session key.*

$$\begin{aligned} a &\rightarrow b : (a, b, enc(pk, p_{a,b})) \\ b &\rightarrow a : enc(enc(k, pk), p_{a,b}) \\ a &\rightarrow b : enc(n_a, k) \\ b &\rightarrow a : enc((n_a, n_b), k) \\ a &\rightarrow b : enc(n_b, k) \end{aligned}$$

Suppose, instead, that pk *is a symmetric key. Can an attacker verify his password guess in some way?*

Example (Cont.)

$$a \rightarrow b : (a, b, enc(pk, p_{a,b}))$$
$$b \rightarrow a : enc(enc(k, pk), p_{a,b})$$
$$a \rightarrow b : enc(n_a, k)$$
$$b \rightarrow a : enc((n_a, n_b), k)$$
$$a \rightarrow b : enc(n_b, k)$$

1. $guess\ p'_{a,b} = 123caas\#$

Example (Cont.)

$$a \to b : (a, b, enc(pk, p_{a,b}))$$
$$b \to a : enc(enc(k, pk), p_{a,b})$$
$$a \to b : enc(n_a, k)$$
$$b \to a : enc((n_a, n_b), k)$$
$$a \to b : enc(n_b, k)$$

1. $guess\ p'_{a,b} = 123caas\#$
2. $enc(k', pk') = dec(enc(enc(k, pk), p_{a,b}), p'_{a,b})$

Example (Cont.)

$$a \rightarrow b : (a, b, enc(pk, p_{a,b}))$$
$$b \rightarrow a : enc(enc(k, pk), p_{a,b})$$
$$a \rightarrow b : enc(n_a, k)$$
$$b \rightarrow a : enc((n_a, n_b), k)$$
$$a \rightarrow b : enc(n_b, k)$$

1. $guess\ p'_{a,b} = 123caas\#$
2. $enc(k', pk') = dec(enc(enc(k, pk), p_{a,b}), p'_{a,b})$
3. $pk' = dec(enc(pk, p_{a,b}), p'_{a,b})$

Example (Cont.)

$$a \rightarrow b : (a, b, enc(pk, p_{a,b}))$$
$$b \rightarrow a : enc(enc(k, pk), p_{a,b})$$
$$a \rightarrow b : enc(n_a, k)$$
$$b \rightarrow a : enc((n_a, n_b), k)$$
$$a \rightarrow b : enc(n_b, k)$$

1. $guess\ p'_{a,b} = 123caas\#$
2. $enc(k', pk') = dec(enc(enc(k, pk), p_{a,b}), p'_{a,b})$
3. $pk' = dec(enc(pk, p_{a,b}), p'_{a,b})$
4. $k' = dec(enc(k', pk'), pk')$

Example (Cont.)

$$a \rightarrow b : (a, b, enc(pk, p_{a,b}))$$
$$b \rightarrow a : enc(enc(k, pk), p_{a,b})$$
$$a \rightarrow b : enc(n_a, k)$$
$$b \rightarrow a : enc((n_a, n_b), k)$$
$$a \rightarrow b : enc(n_b, k)$$

1. $guess\ p'_{a,b} = 123caas\#$
2. $enc(k', pk') = dec(enc(enc(k, pk), p_{a,b}), p'_{a,b})$
3. $pk' = dec(enc(pk, p_{a,b}), p'_{a,b})$
4. $k' = dec(enc(k', pk'), pk')$
5. $n'_a = dec(enc(n_a, k), k')$

Example (Cont.)

$$a \to b : (a, b, enc(pk, p_{a,b}))$$
$$b \to a : enc(enc(k, pk), p_{a,b})$$
$$a \to b : enc(n_a, k)$$
$$b \to a : enc((n_a, n_b), k)$$
$$a \to b : enc(n_b, k)$$

1. $guess\ p'_{a,b} = 123caas\#$
2. $enc(k', pk') = dec(enc(enc(k, pk), p_{a,b}), p'_{a,b})$
3. $pk' = dec(enc(pk, p_{a,b}), p'_{a,b})$
4. $k' = dec(enc(k', pk'), pk')$
5. $n'_a = dec(enc(n_a, k), k')$
6. $n''_a = fst(dec(enc((n_a, n_b), k), k'))$

Example (Cont.)

$$
\begin{aligned}
a &\rightarrow b : (a, b, enc(pk, p_{a,b})) \\
b &\rightarrow a : enc(enc(k, pk), p_{a,b}) \\
a &\rightarrow b : enc(n_a, k) \\
b &\rightarrow a : enc((n_a, n_b), k) \\
a &\rightarrow b : enc(n_b, k)
\end{aligned}
$$

1. $guess\ p'_{a,b} = 123caas\#$
2. $enc(k', pk') = dec(enc(enc(k, pk), p_{a,b}), p'_{a,b})$
3. $pk' = dec(enc(pk, p_{a,b}), p'_{a,b})$
4. $k' = dec(enc(k', pk'), pk')$
5. $n'_a = dec(enc(n_a, k), k')$
6. $n''_a = fst(dec(enc((n_a, n_b), k), k'))$
7. $n'_a \stackrel{?}{=} n''_a$

The protocol is not resistant to off-line gusssing!

Motivation: Wi-Fi Protected Access 2 (WPA2)

Example

Some more modern stuff.

- *Currently, when you are connecting to your access point, you are vulnerable to off-line guessing! An attacker can*
  1. *make a guess*
  2. *get a handshake which contains hash of a password,*
  3. *compare it with its hashed guess.*
- *Wi-Fi Protected Access 3 (WPA3) aims to mitigate this*
  - *It uses a variant of a Dragonfly protocol,*
  - *but, in April 2019, a serious design flow is found [4] which makes the protocol susceptible to an off-line guessing*

Off-line guessing resistance

- When using **weak-secrets** (such as passwords), it is important that an attacker can not verify its guess!
- **Off-line guessing** can be informally defined as the following two-phase game.
  1. **Learning phase.** An attacker can observe and interact with a protocol, learning the messages exchanged during the protocol execution.
  2. **Guessing phase.** An attacker can no longer interact with a protocol (no learning allowed), but it is given a challenge: the weak-secret and a fresh value.
  3. **Verification.** A protocol is off-line guessing resistant is an attacker can not distinguish the weak-secret from a fresh value.
- The learning phase always comes before the guessing phase!
- We can formalize this as a special kind of **behavioral equivalence** between two **labeled transition systems**.

Labeled transition system

Definition (Labeled multiset rewrite system)

*A **labeled multiset rewrite rule** (LMRR) is a tuple* (id, l, a, r)*, written as* $ru = id\colon [l]{-}\![a]\!\rightarrow [r]$*, where* $l, a, r \in \mathcal{F}^{\#}$*, and* $id \in \mathcal{I}$ *is the unique rule identifier. A **labeled multiset rewrite system** (LMRS) is a set of labeled multiset rewrite rules.*

We consider three kinds of LMRS: **system** (Sys), **environment** (Env), and **interface** (IF). The system can interact with the environment using the interface:

$$IF = \begin{cases} out & : \quad [\mathrm{Out}_{sys}(x)]{-}\![\mathrm{O}]\!\rightarrow [\mathrm{In}_{env}(x)], \\ in & : \quad [\mathrm{Out}_{env}(x)]{-}\![\mathrm{I}]\!\rightarrow [\mathrm{In}_{sys}(x)]\ . \end{cases}$$

Example

LMRS S

$$
\begin{aligned}
\mathit{fresh}^{sys}&: [\,]{-}[\,]{\rightarrow}[\mathrm{Fr}(x{:}fr)]\\
\mathit{psgen}^{sys}&: [\mathrm{Fr}(p)]{-}[\mathrm{PL}]{\rightarrow}[!\mathrm{P}(p, a{:}pub, b{:}pub), \mathrm{C}(p)]\\
\mathit{ureq}^{sys}&: [!\mathrm{P}(p, a, b)]{-}[\mathrm{PL}]{\rightarrow}[\mathrm{Out}_{sys}((a, b)), \mathrm{U}(p, a, b)]\\
\mathit{sreq}^{sys}&: [!\mathrm{P}(p, a, b), \mathrm{In}_{sys}((a, b)), \mathrm{Fr}(n)]{-}[\mathrm{PL}()]{\rightarrow}\\
&\ [\mathrm{Out}_{sys}(n), \mathrm{S}(p, a, b, n)]\\
\mathit{ures}^{sys}&: [\mathrm{U}(p, a, b), \mathrm{In}_{sys}(n)]{-}[\mathrm{PL}]{\rightarrow}[\mathrm{Out}_{sys}(\mathrm{enc}(n, p))]\\
\mathit{sver}^{sys}&: [\mathrm{S}(p, a, b, n), \mathrm{In}_{sys}(\mathrm{enc}(n, p))]{-}[\mathrm{PL}]{\rightarrow}[\,]\\
\mathit{chal}^{sys}&: [\mathrm{C}(p), \mathrm{Fr}(f)]{-}[\mathrm{PG}]{\rightarrow}[\mathrm{Out}_{sys}(p)]
\end{aligned}
$$

$$
\begin{aligned}
\mathit{recv}^{env}&: [\mathrm{In}_{env}(x)]{-}[\,]{\rightarrow}[!\mathrm{K}(x)]\\
\mathit{send}^{env}&: [!\mathrm{K}(x)]{-}[\,]{\rightarrow}[\mathrm{Out}_{env}(x)]\\
\mathit{dec}^{env}&: [!\mathrm{K}(\mathrm{enc}(x, y)), !\mathrm{K}(z)]{-}[\,]{\rightarrow}[!\mathrm{K}(\mathrm{dec}(\mathrm{enc}(x, y), z))]\ (\text{modulo } \mathrm{dec}(\mathrm{enc}(x, y), y) = x)\\
\mathit{enc}^{env}&: [!\mathrm{K}(x), !\mathrm{K}(y)]{-}[\,]{\rightarrow}[!\mathrm{K}(\mathrm{enc}(x, y))]\\
\mathit{comp}^{env}&: [!\mathrm{K}(x), !\mathrm{K}(x)]{-}[\,]{\rightarrow}[\,]
\end{aligned}
$$

Protocol

$$
\begin{aligned}
a_{ureq} &\rightarrow b_{sreq} : (a, b)\\
b_{sreq} &\rightarrow a_{ures} : n_b\\
a_{ures} &\rightarrow b_{sver} : enc(n_b, p_{a,b})
\end{aligned}
$$

It aims at to authenticate a to b using a password $p_{a,b}$.

$:fr$	fresh values (keys, passwords,. . .)
$:pub$	public values (names, group generator,. . .)
!	persistent facts (use of public keys, passwords,. . .)
PL, PG	learning and guessing phase

Labeled transition relation

Definition (Labeled transition relation)

Labeled transition relation $\rightarrow_P \subseteq \mathcal{G}^\# \times (\mathcal{G}^\# \times \rho) \times \mathcal{G}^\#$ *of a multiset rewrite system* P *is defined as:*

$$\frac{\begin{array}{c} ri = id\colon [l]\!-\![a]\!\rightarrow\![r] \in_E ginsts(P) \\ lfacts(l) \subseteq^\# S \qquad pfacts(l) \subseteq^\# S \end{array}}{S \xrightarrow[recipe(ri)]{set(a)}_P ((S \setminus^\# lfacts(l)) \cup^\# mset(r))}$$

A multiset rewrite system with a labeled transition relation is called ***labeled transition system*** *(LTS).*

Execution, Trace

Definition (Execution (trace))

*An **execution** e of the multiset rewrite system P is an alternating sequence of states and transitions:*

$$e = [S_0, (l_1 \xrightarrow[rec(ru_1)]{set(a_1)} r_1), S_1, \ldots, S_{k-1}, (l_k \xrightarrow[rec(ru_k)]{set(a_k)} r_k), S_k], S_0 = \emptyset^{\#}.$$

The set of all executions of P is denoted by $exec_P$. Furthermore, we define the following.

$$\begin{aligned} exec_P(S) &= \{e \in exec_P \mid S \text{ is the last state of } e\}, \\ trace(e) &= [set(a_1), \ldots, set(a_k)], \\ trace(R) &= \{trace(e) \mid e \in R\}. \end{aligned}$$

where $R \subseteq exec_P$.

Example

LMRS $S' \subset S$

$$(a_{ureq} \rightarrow b_{sreq} : (a, b))$$

$$\begin{aligned}
fresh^{sys} &: [] \text{-}[]\!\!\rightarrow [\mathrm{Fr}(x{:}fr)] \\
psgen^{sys} &: [\mathrm{Fr}(p)] \text{-}[\mathrm{PL}]\!\!\rightarrow [!\mathrm{P}(p, a{:}pub, b{:}pub), \mathrm{C}(p)] \\
ureq^{sys} &: [!\mathrm{P}(p, a, b)] \text{-}[\mathrm{PL}]\!\!\rightarrow [\mathrm{Out}_{sys}((a, b)), \mathrm{U}(p, a, b)] \\
sreq^{sys} &: [!\mathrm{P}(p, a, b), \mathrm{In}_{sys}((a, b)), \mathrm{Fr}(n)] \text{-}[\mathrm{PL}()]\!\!\rightarrow \\
&\quad [\mathrm{Out}_{sys}(n), \mathrm{S}(p, a, b, n)]
\end{aligned}$$

$$\begin{aligned}
recv^{env} &: [\mathrm{In}_{env}(x)] \text{-}[]\!\!\rightarrow [!\mathrm{K}(x)] \\
send^{env} &: [!\mathrm{K}(x)] \text{-}[]\!\!\rightarrow [\mathrm{Out}_{env}(x)]
\end{aligned}$$

Notice the variable z in $out([], z)$! It specifies that an adversary can not see the internal workings of the system, that is, how the rule *out* was derived.

$$trace(e) = [\mathrm{PL}, \mathrm{O}, \mathrm{I}, \mathrm{PL}]$$

Execution e

$$\emptyset \xrightarrow[fresh([\{^{p_0}/_x\}],[])]{} S_1 = \emptyset \cup^{\#} \{\mathrm{Fr}(p_0)\},$$

$$\xrightarrow[psgen([\{^{a_0}/_a\},\{^{b_1}/_b\}],[fresh_1([\{^{p_0}/_x\}],[])])]{} S_2$$

$$\xrightarrow[ureq([],[psgen_1([\{^{a_0}/_a\},\{^{b_1}/_b\}],[fresh_1([\{^{p_0}/_x\}],[])])])]{\mathrm{PL}} S_3$$

$$\xrightarrow[out([],z)]{\mathrm{O}} S_4$$

$$\xrightarrow[recv([],[out_1([],z])]{} S_5$$

$$\xrightarrow[send([],[recv_1([],[out_1([],z])])]{} S_6$$

$$\xrightarrow[in([],[send_1([],[recv_1([],[out_1([],z])])])]{\mathrm{I}} S_7$$

$$\xrightarrow[fresh([\{^{n_0}/_x\}],[])]{} S_8$$

$$\xrightarrow[sreq([],r_1)]{\mathrm{PL}} S_9$$

$$S_2 = (S_0 \setminus^{\#} \{\mathrm{Fr}(p_0)\}) \cup^{\#} \{!\mathrm{P}(p_0, a_0, b_0), \mathrm{C}(p_0)\},$$

$$S_3 = S_1 \cup^{\#} \{\mathrm{Out}_{sys}((a_0, b_0)), \mathrm{U}(p_0, a_0, b_0)\},$$

$$\vdots$$

Restricted observational equivalence

Definition (Restricted Observational Equivalence)

Two sets of multiset rewrite rules S_A and S_B are **restricted observational equivalent** with respect to the traces $Tr = Tr_A \cup Tr_B$, and an environment given by a set of multiset rewrite rules Env, written as $S_A \approx^{Tr}_{Env} S_B$, if, given the LTS defined by the rules $S_A \cup IF \cup Env$ and $S_B \cup IF \cup Env$, there exist a relation $\mathcal{R}$ containing the initial states, such that for all states $(\mathcal{S}_A, \mathcal{S}_B) \in \mathcal{R}$ the following conditions hold.

Restricted observational equivalence

Definition (Cont.)

1. There exist traces $tr_A \in trace(execs_{S_A}(\mathcal{S}_A))$ and $tr_B \in trace(execs_{S_B}(\mathcal{S}_B))$ such that $tr_A \in Tr_A$ and $tr_B \in Tr_B$.
2. If $\mathcal{S}_A \xrightarrow[r]{l} \mathcal{S}'_A$ and $concat(tr_A, [l]) \in Tr_A$ where r is the recipe of a rule in $Env \cup IF$ then there exist $l' \in \mathcal{F}^{\#}$ and $\mathcal{S}'_B \in \mathcal{G}^{\#}$ such that $\mathcal{S}_B \xrightarrow[r]{l'} \mathcal{S}'_B$, $concat(tr_B, [l']) \in Tr_B$, and $(\mathcal{S}'_A, \mathcal{S}'_B) \in \mathcal{R}$.
3. If $\mathcal{S}_A \xrightarrow[r]{l} \mathcal{S}'_A$ and $concat(tr_A, [l]) \in Tr_A$ where r is the recipe of a rule in S_A then there exist recipes $r_1, \ldots, r_n \in \rho$ of rules in S_B, actions $l_1, \ldots, l_n \in \mathcal{F}^{\#}$, $n \geq 0$, and $\mathcal{S}'_B \in \mathcal{G}^{\#}$ such that $\mathcal{S}_B \xrightarrow[r_1]{l_1} \ldots \xrightarrow[r_n]{l_n} \mathcal{S}'_B$, $concat(tr_B, [l_1, \ldots, l_n]) \in Tr_B$, and $(\mathcal{S}'_A, \mathcal{S}'_B) \in \mathcal{R}$.

Conditions 2 and 3 must also hold in the other direction!

Restricted observational equivalence: the motivation

This gives us the ability to fine-grain observational equivalence, e.g., to only consider the traces of our interest.

- action ordering
  $\forall\ ptr\ \#i\ \#j.\ read(ptr)@i \wedge free(ptr)@j \rightarrow \#i < \#j$
- (in)equality checks
  $\forall\ x\ y\ \#i.\ equal(x, y)@i \rightarrow x = y$
- forcing uniqueness
  $\forall\ x\ \#i\ \#j.\ uniq(x)@i \wedge uniq(x)@j \rightarrow \#i = \#j$
- disallowing certain conditions
  $\forall\ x\ \#i.\ bad(x)@i \rightarrow \perp$

Verification

Verifying observational equivalence is generally undecidable. Automation is difficult. Some of the difficulties come from the multiset rewriting semantics, e.g.,

- execution does not contain casual dependencies between steps,
- execution can contain redundant steps,

while other from the (restricted) observational equivalence itself, e.g.,

- a rule can have many different recipes,
- an internal step can be simulated by an arbitrary number of steps on the other side

but, we can make things easier with **bi-systems** and **(partial) dependency graphs**.

Bi-system

Again, we want to see if the two LMRS behave the same: the one that gives us the **weak-secret**, and the other that gives us a **random value**. We can implicitly specify two systems by a bi-system.

Definition

*A **bi-system** S is a LMRS with* diff(_ , _) *operator such that a pair of LMRS L(S) and R(S) can be obtained by considering the left hand side (LHS) and the right hand side (RHS) of* diff(_ , _) *operator respectively.*

Example

We can construct the bi-system B from the system S by modifiying the challenge rule:

$$chal^{sys}\colon [\mathrm{C}(p{:}fr), \mathrm{Fr}(f{:}fr)] \!-\![\mathrm{PG}]\!\rightarrow [\mathrm{Out}_{sys}(\mathrm{diff}(p{:}fr, f{:}fr)] \; .$$

Example (prev.)

LMRS S

$$
\begin{aligned}
\mathit{fresh}^{sys}&: [\,]\!-\![\,]\!\rightarrow\![\mathrm{Fr}(x{:}fr)]\\
\mathit{psgen}^{sys}&: [\mathrm{Fr}(p)]\!-\![\mathrm{PL}]\!\rightarrow\![!\mathrm{P}(p, a{:}pub, b{:}pub), \mathrm{C}(p)]\\
\mathit{ureq}^{sys}&: [!\mathrm{P}(p,a,b)]\!-\![\mathrm{PL}]\!\rightarrow\![\mathrm{Out}_{sys}((a,b)), \mathrm{U}(p,a,b)]\\
\mathit{sreq}^{sys}&: [!\mathrm{P}(p,a,b), \mathrm{In}_{sys}((a,b)), \mathrm{Fr}(n)]\!-\![\mathrm{PL}()]\!\rightarrow\\
&\quad [\mathrm{Out}_{sys}(n), \mathrm{S}(p,a,b,n)]\\
\mathit{ures}^{sys}&: [\mathrm{U}(p,a,b), \mathrm{In}_{sys}(n)]\!-\![\mathrm{PL}]\!\rightarrow\![\mathrm{Out}_{sys}(\mathrm{enc}(n,p))]\\
\mathit{sver}^{sys}&: [\mathrm{S}(p,a,b,n), \mathrm{In}_{sys}(\mathrm{enc}(n,p))]\!-\![\mathrm{PL}]\!\rightarrow\![\,]\\
\mathit{chal}^{sys}&: [\mathrm{C}(p), \mathrm{Fr}(f)]\!-\![\mathrm{PG}]\!\rightarrow\![\mathrm{Out}_{sys}(\mathrm{diff}(p,f))]
\end{aligned}
$$

$$
\begin{aligned}
\mathit{recv}^{env}&: [\mathrm{In}_{env}(x)]\!-\![\,]\!\rightarrow\![!\mathrm{K}(x)]\\
\mathit{send}^{env}&: [!\mathrm{K}(x)]\!-\![\,]\!\rightarrow\![\mathrm{Out}_{env}(x)]\\
\mathit{dec}^{env}&: [!\mathrm{K}(\mathrm{enc}(x,y)), !\mathrm{K}(z)]\!-\![\,]\!\rightarrow\![!\mathrm{K}(\mathrm{dec}(\mathrm{enc}(x,y),z))]\ \ (\text{modulo } \mathrm{dec}(\mathrm{enc}(x,y),y) = x)\\
\mathit{enc}^{env}&: [!\mathrm{K}(x), !\mathrm{K}(y)]\!-\![\,]\!\rightarrow\![!\mathrm{K}(\mathrm{enc}(x,y))]\\
\mathit{comp}^{env}&: [!\mathrm{K}(x), !\mathrm{K}(x)]\!-\![\,]\!\rightarrow\![\,]
\end{aligned}
$$

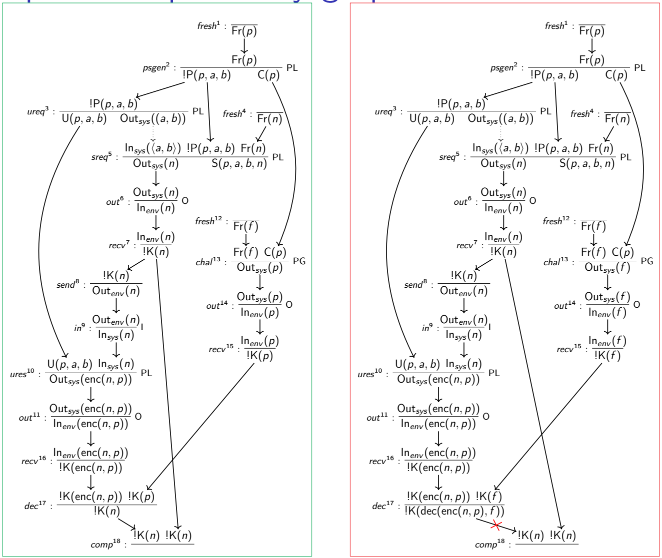
Protocol

$$
\begin{aligned}
a_{ureq} &\rightarrow b_{sreq} : (a, b)\\
b_{sreq} &\rightarrow a_{ures} : n_b\\
a_{ures} &\rightarrow b_{sver} : enc(n_b, p_{a,b})
\end{aligned}
$$

It aims at to authenticate a to b using a password $p_{a,b}$.

$:fr$	fresh values (keys, passwords,...)
$:pub$	public values (names, group generator,...)
!	persistent facts (use of public keys, passwords,...)
PL, PG	learning and guessing phase

Example: partial dependency graphs



$$tr = [\mathrm{PL}, \mathrm{PL}, \mathrm{O}, \mathrm{I}, \mathrm{PL}, \mathrm{O}, \mathrm{I}, \mathrm{PL}, \mathrm{O}, \mathrm{PG}, \mathrm{O}] \ \checkmark$$

Partial dependency graph

Definition (Partial Dependency Graph)

Let E be an equational theory, R a set of labeled multiset rewrite protocol rules, Env an environment. We say that a pair $pdg = (I, D)$ is a **(partial) dependency graph** (PDG) modulo E for R, if $I \in_E (ginsts(R \cup IF \cup Env))^*$, $D \subseteq \mathbb{N}^2 \times \mathbb{N}^2$ and the following holds.

1. For every edge $(i, u) \rightarrowtail (j, v) \in D$, it holds that $i < j$ and $concs(I_i)_u =_E prems(I_j)_v$.
2. Every premise of pdg has (at most) one incoming edge.
3. Every linear conclusion of pdg has at most one outgoing edge.
4. The Fresh instances are unique.

Lemma (we do not loose anything [3])

For all proto. P, $trace(exec(P \cup Env)) = trace(dgraphs(P \cup Env))$

Partial dependency graph equivalence

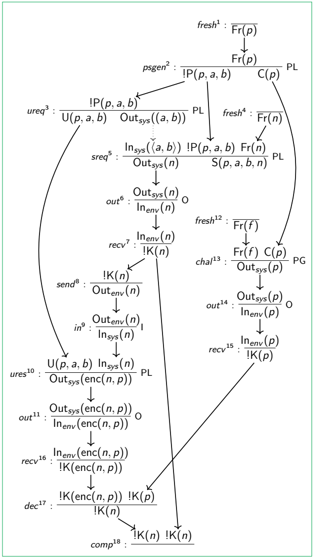
Graphs naturally give rise to an equivalence relation.

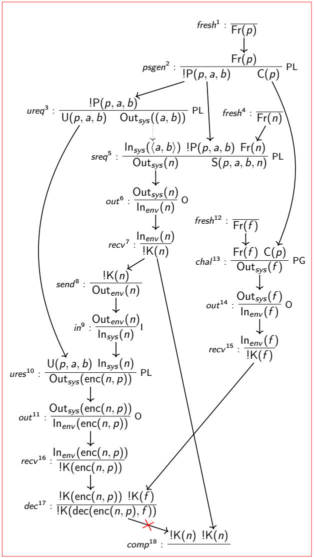
Definition (Partial dependency graph equivalence)

Let R be a set of labeled multiset rewrite protocol rules and Env an environment. For a $S = pdgraphs(R \cup Env \cup IF)$, we say that the partial dependency graphs $pdg = (I, D) \in S$ and $pdg' = (I', D') \in S$, are **equivalent**, written as $pdg \simeq_S pdg'$, if $D = D'$, $|I| = |I'|$, $idx(I) = idx(I')$, and for all $i \in idx(I)$ it holds that I_i and I'_i are ground instances of the same rules.

The relation $\simeq_S$ is an **equivalence relation**, and $[pdg] \in S/\simeq$ is a **partial dependency graph class**.

Example: equivalence





These are not equivalent!

Partial dependency graph mirror

Definition (Partial dependency graph mirror)

Let S be a protocol bi-system, Env an environment, $L = L(S) \cup IF \cup Env$ and $R = R(S) \cup IF \cup Env$ corresponding LMRS, and $G = pdgraphs(L) \cup pdgraphs(R)$. For $pdg_L \in pdgraphs(L)$, we define

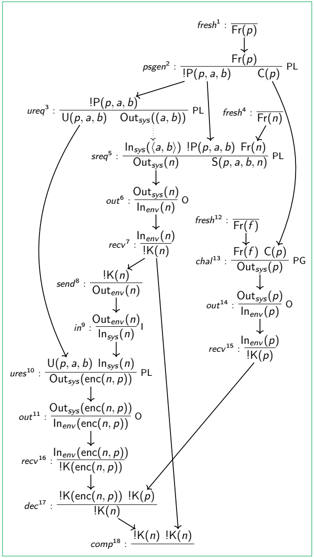
$$mirror(pdg_L) = \{pdg_R \in pdgraphs(R) \mid pdg_R \simeq_G pdg_L\},$$

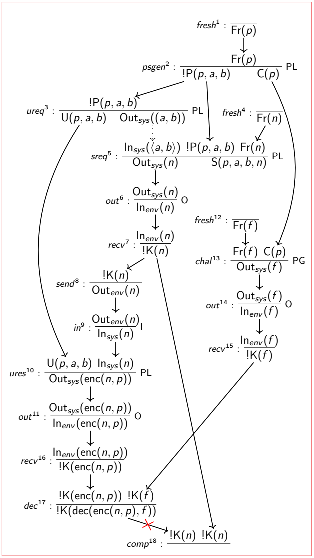
and, for $[pdg_L] \in pdgraphs(L)/\simeq$, we define

$$mirror([pdg_L]) = \bigcup_{pdg \in [pdg_L]} mirror(pdg).$$

We define analogously in the other direction.

Example: partial dependency graph mirror





LHS does not have a mirror!

Restricted Dependency Graph Equivalence

Definition (System restricted dependency graph equivalence)

Let S be a bi-system and consider multiset rewrite systems $L = L(S) \cup IF \cup Env$ and $R = R(S) \cup IF \cup Env$. For a set of traces $Tr = Tr_A \cup Tr_B$ we say that L and R are **restricted dependency graph equivalent** written as $L(S) \sim^{Tr}_{DG,Env} R(S)$, if $dg \in dgraphs(L \cup R)$ such that $trace(dg) \in Tr$, implies $mirror(dg) \neq \emptyset$, and for all $dg' \in mirror(dg)$ it holds that $trace(dg') \in Tr$.

Theorem (Sound approximation)

Let S be a bi-system and $Tr = Tr_L \cup Tr_R$ be a set of traces. If $L(S) \sim^{Tr}_{DG,Env} R(S)$ then $L(S) \approx^{Tr}_{Env} R(S)$.

Tamarin prover

- Restricted dependency graph equivalence makes verification easier, but a lot more constraints are needed for a partial automation: protocol rules, adversary rules, equational theories, first-order formulas. . .
- We take existing security protocol verification tool called **Tamarin prover** [2] and naturally extend its observational equivalence [1] to include well-defined restrictions[1].
- Our extension can prove restricted observational equivalence for a simple class of safety properties that only depend on the structure of the execution, but not on the data:

$$\forall \#i \ \#j. \ \mathrm{PhaseLearn}() \wedge \mathrm{PhaseGuess}() \rightarrow \#i < \#j.$$

- We use this to prove off-line guessing resistance of **Encrypted Key Exchange** (EAP-EKE) protocol

[1] Tamarin already supported restrictions, but to the best of our knowledge, the semantics and the soundness of the proof method were missing

References

[1] David Basin, Jannik Dreier, and Ralf Sasse. 2015. Automated Symbolic Proofs of Observational Equivalence. *In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. Association for Computing Machinery, New York, NY, USA, 1144–1155. DOI: `https://doi.org/10.1145/2810103.2813662`

[2] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse, Benedikt Schmidt, and contributors, *The Tamarin Prover tool*, GitHub repository (accessed on September 2021): `https://github.com/tamarin-prover/tamarin-prover`

[3] B. Schmidt, S. Meier, C. Cremers and D. Basin, "Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties," 2012 IEEE 25th Computer Security Foundations Symposium, 2012, pp. 78-94, doi: 10.1109/CSF.2012.25.

[4] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd," 2020 IEEE Symposium on Security and Privacy (SP), 2020, pp. 517-533, doi: 10.1109/SP40000.2020.00031.