Automated Verification of Equivalence Properties of Cryptographic Protocols

Rohit Chadha[1] Ştefan Ciobâcă[2] Steve Kremer[3]

[1] LSV & INRIA Saclay, France [2] University "Al. I. Cuza", Iasi, Romania

[3] LORIA & INRIA Nancy, France

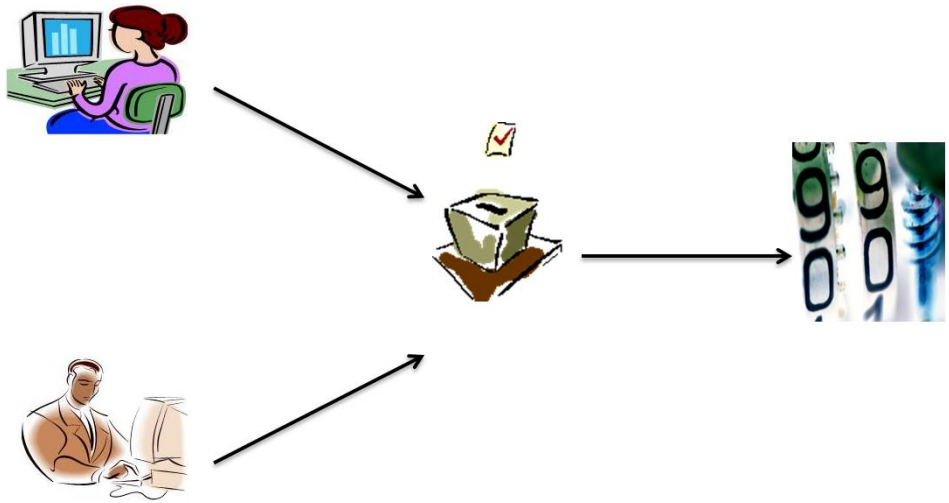
ESOP' 12

Electronic (Internet) voting

Desired security goals

Eligibility: Only eligible voters voted

Privacy: Nobody should learn the choice of a particular voter



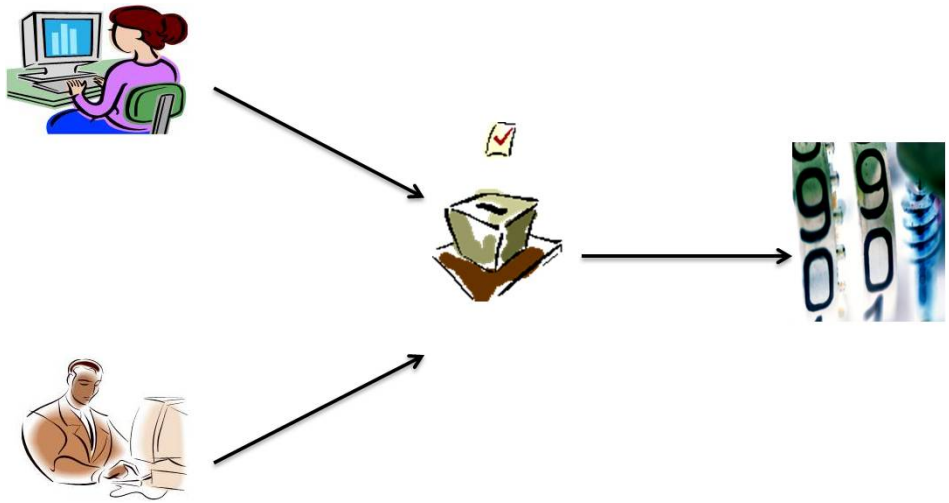
Electronic (Internet) voting

Desired security goals

Eligibility: Only eligible voters voted

Privacy: Nobody should learn the choice of a particular voter

Is it enough to send the votes encrypted?

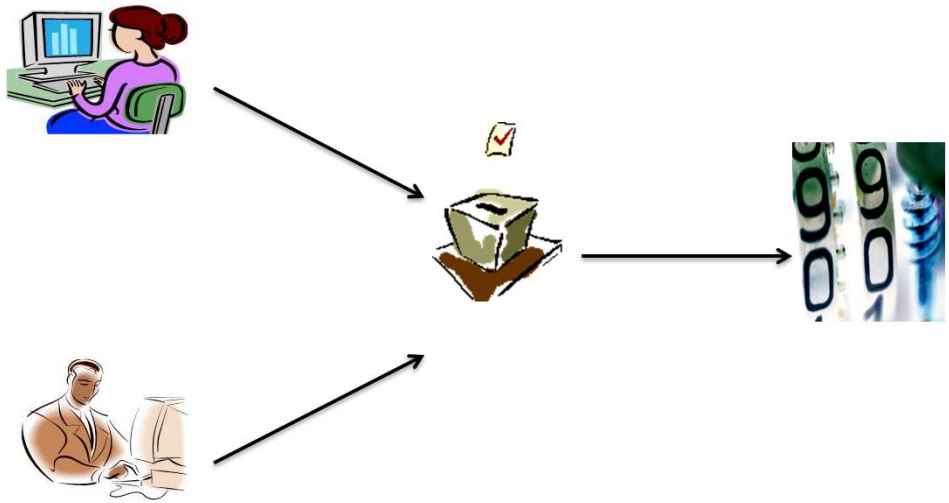


Electronic (Internet) voting

Desired security goals

Eligibility: Only eligible voters voted

Privacy: Nobody should learn the choice of a particular voter



Is it enough to send the votes encrypted?

What if the votes are tallied in the order they are received?

Privacy as indistinguishability (equivalence)

Privacy in electronic voting

The attacker by interacting with the protocol should not be able to distinguish between the following two scenarios:

1. Honest voter Alice votes for choice 0 and honest voter Bob votes for choice 1
2. Honest voter Alice votes for choice 1 and honest voter Bob votes for choice 0

Indistinguishability is key

Indistinguishability is essential for modeling important security properties

- Privacy in electronic voting
- Resistance against offline guessing attacks (real-or-random): After the protocol executes, the attacker when presented with the real secret and a random sting cannot tell which one is the secret
- Unlinkability in RFID systems: Multiple uses of the system by the same user cannot be linked together
- and many others: strong secrecy, abuse-freeness in contract-signing, ...

Existing work on verifying indistinguishability properties

- NP completeness results for indistinguishability of two symbolic traces [Baudet'05,Chevalier & Rusinowitch'10]

  $\rightsquigarrow$ allows to verify trace equivalence for a class of simple processes for a bounded number of sessions [Cortier & Delaune'10]

  $\rightsquigarrow$ procedures are highly non-deterministic and not reasonably implementable;

- more practical procedures [Cheval, Comon-Lundh, Delaune '10,'11, Dawson & Tiu'10]

  $\rightsquigarrow$ restricted support of cryptographic primitives (encryption, signatures, hash)

- indistinguishability verified by ProVerif [Blanchet, Abadi & Fournet'05]

  $\rightsquigarrow$ efficient procedure for an unbounded number of sessions, but due to approximations proofs fail for interesting protocols

Our tool for verifying indistinguishability properties

[Chadha,Ciobaca,Kremer, ESOP' 2012]

AKiSs

Active Knowledge In Security protocolS

`http://www.lsv.ens-cachan.fr/~ciobaca/akiss/`

$\sim$2000 lines of OCaml code

Our tool for verifying indistinguishability properties

[Chadha,Ciobaca,Kremer, ESOP' 2012]

AKiSs

Active Knowledge In Security protocolS

`http://www.lsv.ens-cachan.fr/~ciobaca/akiss/`

$\sim$2000 lines of OCaml code

- Vote privacy for FOO and Okamoto electronic voting protocols
  First automated proof

Our tool for verifying indistinguishability properties

[Chadha,Ciobaca,Kremer, ESOP' 2012]

AKiSs

Active Knowledge In Security protocolS

`http://www.lsv.ens-cachan.fr/~ciobaca/akiss/`

$\sim$2000 lines of OCaml code

- Vote privacy for FOO and Okamoto electronic voting protocols
  First automated proof
- Strong secrecy: NSL protocol and Blanchet's variant's of Denning-Sacco
- Resistance to offline guessing attacks: EKE

What does AKiSs do?

What is indistinguishability?

- Model of cryptographic operations
- Protocol execution model
- Definition of indistinguishability

What does AKiSs do?

What is indistinguishability?

- Model of cryptographic operations
- Protocol execution model
- Definition of indistinguishability

What techniques we employ?

- Logic programming (Horn-clause resolution)
- Rewriting techniques (most general unifiers, variants)

Modeling (perfect) cryptographic operations

- Messages are modeled as first-order symbolic terms (not bitstrings)
  - The term $\mathsf{senc}(k, m, r)$ models the symbolic encryption of plain-text m under key k with randomness r
- Secret values are modeled as names in a set $\mathcal{N}$. The secret is not known to the adversary unless leaked by the protocol.
- Equations between terms model cryptographic operations
  - $\mathsf{sdec}(x, \mathsf{senc}(x, y, z)) = y$

Modeling (perfect) cryptographic operations

- Messages are modeled as first-order symbolic terms (not bitstrings)
  - The term $\mathsf{senc}(k, m, r)$ models the symbolic encryption of plain-text m under key k with randomness r
- Secret values are modeled as names in a set $\mathcal{N}$. The secret is not known to the adversary unless leaked by the protocol.
- Equations between terms model cryptographic operations
  - $\mathsf{sdec}(x, \mathsf{senc}(x, y, z)) = y$

Need to handle symbolic equations

Modeling protocol execution

- Protocol execution is modeled as a labeled transition system
  - The states consists of local states of the protocol participants and the attacker state
  - The labels model communication with the attacker

Modeling protocol execution

- Protocol execution is modeled as a labeled transition system
  - The states consists of local states of the protocol participants and the attacker state
  - The labels model communication with the attacker
- The attacker state (called frame) is the sequence of messages it has received so far
  - Example: $\varphi = \{^{\mathsf{senc}(k_1,m_1,r_1)}/_{w_1}, ^{\mathsf{senc}(k_2,m_2,r_2)}/_{w_2}, ^{k_1}/_{w_3}\}$

Modeling protocol execution

- Protocol execution is modeled as a labeled transition system
  - The states consists of local states of the protocol participants and the attacker state
  - The labels model communication with the attacker
- The attacker state (called frame) is the sequence of messages it has received so far
  - Example: $\varphi = \{^{\mathsf{senc}(k_1,m_1,r_1)}/_{w_1}, ^{\mathsf{senc}(k_2,m_2,r_2)}/_{w_2}, ^{k_1}/_{w_3}\}$
  - Attacker can compute (deduce) new terms from frames when sending messges (Potentially infinite)
    Example:
    $\mathsf{sdec}(w_3, w_1), \mathsf{sdec}(w_2, w_3), \mathsf{sdec}(w_2, w_3), \mathsf{senc}(\mathsf{sdec}(w_1, w_3, r_3), w_2), \ldots$

Modeling protocol execution

- Protocol execution is modeled as a labeled transition system
  - The states consists of local states of the protocol participants and the attacker state
  - The labels model communication with the attacker
- The attacker state (called frame) is the sequence of messages it has received so far
  - Example: $\varphi = \{^{\mathsf{senc}(k_1,m_1,r_1)}/_{w_1}, ^{\mathsf{senc}(k_2,m_2,r_2)}/_{w_2}, ^{k_1}/_{w_3}\}$
  - Attacker can compute (deduce) new terms from frames when sending messges (Potentially infinite)
    Example:
    $\mathsf{sdec}(w_3, w_1), \mathsf{sdec}(w_2, w_3), \mathsf{sdec}(w_2, w_3), \mathsf{senc}(\mathsf{sdec}(w_1, w_3, r_3), w_2), \ldots$

Need to handle infinite-state system

Indistinguishability

- Indistinguishability is trace equivalence and is defined by
  - First defining a static equivalence on frames, i.e, the attacker states
  - Lift static equivalence to traces.

Indistinguishability

- Indistinguishability is trace equivalence and is defined by
  - First defining a static equivalence on frames, i.e, the attacker states
    Two frames are equivalent iff the same equations hold in them

$$\{^{\mathsf{senc}(k,\mathbf{0},r_0)}/_{w_1}\} \quad \approx \quad \{^{\mathsf{senc}(k,\mathbf{1},r_1)}/_{w_1}\}$$
$$\{^{\mathsf{senc}(k,\mathbf{0},r_0)}/_{w_1}, {}^{k}/_{w_2}\} \quad \not\approx \quad \{^{\mathsf{senc}(k,\mathbf{1},r_1)}/_{w_1}, {}^{k}/_{w_2}\} \quad (\mathsf{sdec}(w_2, w_1) \stackrel{?}{=} 0)$$

  - Lift static equivalence to traces.

Indistinguishability

- Indistinguishability is trace equivalence and is defined by
  - First defining a static equivalence on frames, i.e, the attacker states
    Two frames are equivalent iff the same equations hold in them

$$\{^{\mathsf{senc}(k,\mathbf{0},r_0)}/_{w_1}\} \quad \approx \quad \{^{\mathsf{senc}(k,\mathbf{1},r_1)}/_{w_1}\}$$
$$\{^{\mathsf{senc}(k,\mathbf{0},r_0)}/_{w_1}, {}^{k}/_{w_2}\} \quad \not\approx \quad \{^{\mathsf{senc}(k,\mathbf{1},r_1)}/_{w_1}, {}^{k}/_{w_2}\} \quad (\mathsf{sdec}(w_2, w_1) \stackrel{?}{=} 0)$$

  - Lift static equivalence to traces.

Need to check static equivalence

Our approach

Rewriting to handle equations

- Handle equational theories that can be oriented into a (optimally reducing) convergent rewrite system

$$\mathsf{sdec}(x, \mathsf{senc}(x, y, z)) \to y$$

- Effects of rewriting can be pre-computed using invariants, eg, $\mathsf{sdec}(x, t)$ can possibly rewrite to y if t is $\mathsf{senc}(x, y, r)$

Our approach

Rewriting to handle equations

- Handle equational theories that can be oriented into a (optimally reducing) convergent rewrite system

$$\mathsf{sdec}(x, \mathsf{senc}(x, y, z)) \to y$$

- Effects of rewriting can be pre-computed using invariants, eg, $\mathsf{sdec}(x, t)$ can possibly rewrite to y if t is $\mathsf{senc}(x, y, r)$

Logic Programming

- Represent traces in Horn-clauses
- Can handle infinite states

$$\mathsf{AttackerKnows}(\mathsf{senc}(x, y)) \Leftarrow \mathsf{AttackerKnows}(x), \mathsf{AttackerKnows}(y)$$

Formal model

Terms and frames

Messages are modelled as first-order terms equipped with a convergent rewrite system R.

Secret values are modelled as names in a set $\mathcal{N}$.

We write $t =_R u$ when $t{\downarrow} = u{\downarrow}$

Sequences of messages are grouped in a frame $\varphi = \{{}^{t_1}/_{w_1}, \ldots, {}^{t_n}/_{w_n}\}$

Terms and frames

Messages are modelled as first-order terms equipped with a convergent rewrite system R.

Secret values are modelled as names in a set $\mathcal{N}$.

We write $t =_R u$ when $t{\downarrow} = u{\downarrow}$

Example

Vocabulary: senc/3, sdec/2
Rewrite system: $\mathrm{sdec}(\mathrm{senc}(x, y, z), y) \to_R x$
Terms: $t_1 = \mathrm{senc}(n, k, r)$, $t_2 = \mathrm{sdec}(t_1, k)$ $\quad (n, k, r \in \mathcal{N})$
We have that $t_2 =_R n$

Sequences of messages are grouped in a frame $\varphi = \{{}^{t_1}/_{w_1}, \ldots, {}^{t_n}/_{w_n}\}$

Deduction

What messages can an attacker compute?

Definition (Deduction)

A term t is *deducible from frame* φ *with a recipe* r ($\varphi \vdash^r t$) if $r\varphi =_R t$ and r does not contain names in $\mathcal{N}$.

Example

Let $\varphi = \{^{\mathsf{senc}(n_1,k_1,r_1)}/_{w_1}, ^{\mathsf{senc}(n_2,k_2,r_2)}/_{w_2}, ^{k_1}/_{w_3}\}$.

We have that $\varphi \vdash^{\mathsf{sdec}(w_1,w_3)} n_1$, $\varphi \not\vdash n_2$

Static equivalence

Indistinguishability of sequences of messages

Definition (Static equivalence)

$(r_1 = r_2)\varphi$ if $\varphi \vdash^{r_1} t$ and $\varphi \vdash^{r_2} t$ for some t.

φ_1 *statically included* in φ_2 ($\varphi_1 \sqsubseteq_s \varphi_2$) iff $(r_1 = r_2)\varphi_1$ implies $(r_1 = r_2)\varphi_2$.

φ_1 and φ_2 are *statically equivalent* ($\varphi_1 \approx_s \varphi_2$) iff $\varphi_1 \sqsubseteq_s \varphi_2$ and $\varphi_2 \sqsubseteq_s \varphi_1$.

Examples

$$\left\{{}^{n_1}/_{w_1}, {}^{n_2}/_{w_2}\right\} \not\approx_s \left\{{}^{n_1}/_{w_1}, {}^{n_1}/_{w_2}\right\} \qquad (w_1 \stackrel{?}{=} w_2)$$

$$\left\{{}^{n_1}/_{w_1}, {}^{n_2}/_{w_2}\right\} \sqsubseteq_s \left\{{}^{n_1}/_{w_1}, {}^{n_1}/_{w_2}\right\}$$

$$\left\{{}^{\mathsf{senc}(\mathbf{0},k,r)}/_{w_1}\right\} \approx_s \left\{{}^{\mathsf{senc}(\mathbf{1},k,r)}/_{w_1}\right\}$$

$$\left\{{}^{\mathsf{senc}(n,k,r)}/_{w_1}, {}^{k}/_{w_2}\right\} \not\approx_s \left\{{}^{\mathsf{senc}(\mathbf{0},k,r)}/_{w_1}, {}^{k}/_{w_2}\right\} \qquad (sdec(w_1, w_2) \stackrel{?}{=} \mathbf{0})$$

A simple crypto process calculus

Actions : $\mathsf{receive}(c, x) \mid \mathsf{send}(c, t) \mid [s \stackrel{?}{=} t]$

Trace: sequence of actions

Process: set of traces

Operational semantics: $(T, \varphi) \xrightarrow{\ell} (T', \varphi')$

$$\textsf{Receive}\ \frac{\varphi \vdash^{r} t}{(\mathsf{receive}(c, x).T, \varphi) \xrightarrow{\mathsf{receive}(c,r)} (T\{x \mapsto t\}, \varphi)}$$

$$\textsf{Test}\ \frac{s =_{\mathsf{R}} t}{([s \stackrel{?}{=} t].T, \varphi) \xrightarrow{\mathsf{test}} (T, \varphi)}$$

$$\textsf{Send}\ \frac{}{(\mathsf{send}(c, t).T, \varphi) \xrightarrow{\mathsf{send}(c)} (T, \varphi \cup \{w_{|dom(\varphi)|+1} \mapsto t\})}$$

Trace equivalences

Trace equivalence

$P \sqsubseteq_t Q$ if $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (P', \varphi)$ implies $\exists Q', \varphi'.\ (Q, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (Q', \varphi') \wedge \varphi \sim_s \varphi'$.

$P \approx_t Q$ iff $P \sqsubseteq_t Q \wedge Q \sqsubseteq_t P$.

Trace equivalences

Fine grained trace equivalence

$P \sqsubseteq_{ft} Q$ if $\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

$P \approx_{ft} Q$ iff $P \sqsubseteq_{ft} Q \wedge Q \sqsubseteq_{ft} P$.

Trace equivalence

$P \sqsubseteq_t Q$ if $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (P', \varphi)$ implies

$\exists Q', \varphi'.\ (Q, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (Q', \varphi') \wedge \varphi \sim_s \varphi'$.

$P \approx_t Q$ iff $P \sqsubseteq_t Q \wedge Q \sqsubseteq_t P$.

Trace equivalences

Fine grained trace equivalence

$P \sqsubseteq_{ft} Q$ if $\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

$P \approx_{ft} Q$ iff $P \sqsubseteq_{ft} Q \wedge Q \sqsubseteq_{ft} P$.

Trace equivalence

$P \sqsubseteq_t Q$ if $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (P', \varphi)$ implies $\exists Q', \varphi'.\ (Q, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (Q', \varphi') \wedge \varphi \sim_s \varphi'$.

$P \approx_t Q$ iff $P \sqsubseteq_t Q \wedge Q \sqsubseteq_t P$.

Example:

$P \approx_t Q$ but $P \not\approx_{ft} Q$

$P = \{\mathsf{send}(c, enc(a,k)).\ \mathsf{send}(c, enc(b,k)).\ \mathsf{receive}(c,x).\ [x = enc(a,k)].\mathsf{send}(c,k),$
$\mathsf{send}(c, enc(a,k)).\ \mathsf{send}(c, enc(b,k)).\ \mathsf{receive}(c,x).\ [x = enc(b,k)].\mathsf{send}(c,k)\}$

$Q = \{\mathsf{send}(c, enc(a,k)).\ \mathsf{send}(c, enc(b,k)).\ \mathsf{receive}(c,x).\ [x = enc(dec(x,k),k)].\ \mathsf{send}(c,k)\}$

Trace equivalences

Fine grained trace equivalence

$P \sqsubseteq_{ft} Q$ if $\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

$P \approx_{ft} Q$ iff $P \sqsubseteq_{ft} Q \wedge Q \sqsubseteq_{ft} P$.

Trace equivalence

$P \sqsubseteq_t Q$ if $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (P', \varphi)$ implies

$\exists Q', \varphi'.\ (Q, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (Q', \varphi') \wedge \varphi \sim_s \varphi'$.

$P \approx_t Q$ iff $P \sqsubseteq_t Q \wedge Q \sqsubseteq_t P$.

Coarse trace equivalence

$P \sqsubseteq_{ct} Q$ if $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (P', \varphi)$ implies

$\exists Q', \varphi'.\ (Q, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (Q', \varphi') \wedge \varphi \sqsubseteq_s \varphi'$.

$P \approx_{ct} Q$ iff $P \sqsubseteq_{ct} Q \wedge Q \sqsubseteq_{ct} P$.

Trace equivalences

Fine grained trace equivalence

$P \sqsubseteq_{ft} Q$ if $\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

$P \approx_{ft} Q$ iff $P \sqsubseteq_{ft} Q \wedge Q \sqsubseteq_{ft} P$.

Trace equivalence

$P \sqsubseteq_t Q$ if $(P, \emptyset) \xRightarrow{\ell_1,\ldots,\ell_n} (P', \varphi)$ implies

$\exists Q', \varphi'.\ (Q, \emptyset) \xRightarrow{\ell_1,\ldots,\ell_n} (Q', \varphi') \wedge \varphi \sim_s \varphi'$.

$P \approx_t Q$ iff $P \sqsubseteq_t Q \wedge Q \sqsubseteq_t P$.

Coarse trace equivalence

$P \sqsubseteq_{ct} Q$ if $(P, \emptyset) \xRightarrow{\ell_1,\ldots,\ell_n} (P', \varphi)$ implies

$\exists Q', \varphi'.\ (Q, \emptyset) \xRightarrow{\ell_1,\ldots,\ell_n} (Q', \varphi') \wedge \varphi \sqsubseteq_s \varphi'$.

$P \approx_{ct} Q$ iff $P \sqsubseteq_{ct} Q \wedge Q \sqsubseteq_{ct} P$.

Example:

$P \approx_{ct} Q$ but $P \not\approx_t Q$

$P = \{\mathsf{send}(c, a).\mathsf{send}(c, a)\}$

$Q = \{\mathsf{send}(c, a).\mathsf{send}(c, a),$
$\mathsf{send}(c, a).\mathsf{send}(c, b)\}$.

Trace equivalences

Fine grained trace equivalence

$P \sqsubseteq_{ft} Q$ if $\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

$P \approx_{ft} Q$ iff $P \sqsubseteq_{ft} Q \wedge Q \sqsubseteq_{ft} P$.

Trace equivalence

$P \sqsubseteq_t Q$ if $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (P', \varphi)$ implies

$\exists Q', \varphi'.\ (Q, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (Q', \varphi') \wedge \varphi \sim_s \varphi'$.

$P \approx_t Q$ iff $P \sqsubseteq_t Q \wedge Q \sqsubseteq_t P$.

Coarse trace equivalence

$P \sqsubseteq_{ct} Q$ if $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (P', \varphi)$ implies

$\exists Q', \varphi'.\ (Q, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (Q', \varphi') \wedge \varphi \sqsubseteq_s \varphi'$.

$P \approx_{ct} Q$ iff $P \sqsubseteq_{ct} Q \wedge Q \sqsubseteq_{ct} P$.

Definition:

P is *determinate* if whenever $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (T, \varphi)$ and $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (T', \varphi')$ then $\varphi \approx_s \varphi'$.

Trace equivalences

Fine grained trace equivalence

$P \sqsubseteq_{ft} Q$ if $\forall T \in P.\ \exists T' \in Q.\ T \approx_t T'$

$P \approx_{ft} Q$ iff $P \sqsubseteq_{ft} Q \wedge Q \sqsubseteq_{ft} P$.

Trace equivalence

$P \sqsubseteq_t Q$ if $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (P', \varphi)$ implies

$\exists Q', \varphi'.\ (Q, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (Q', \varphi') \wedge \varphi \sim_s \varphi'$.

$P \approx_t Q$ iff $P \sqsubseteq_t Q \wedge Q \sqsubseteq_t P$.

det. proc.

Coarse trace equivalence

$P \sqsubseteq_{ct} Q$ if $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (P', \varphi)$ implies

$\exists Q', \varphi'.\ (Q, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (Q', \varphi') \wedge \varphi \sqsubseteq_s \varphi'$.

$P \approx_{ct} Q$ iff $P \sqsubseteq_{ct} Q \wedge Q \sqsubseteq_{ct} P$.

Remark:

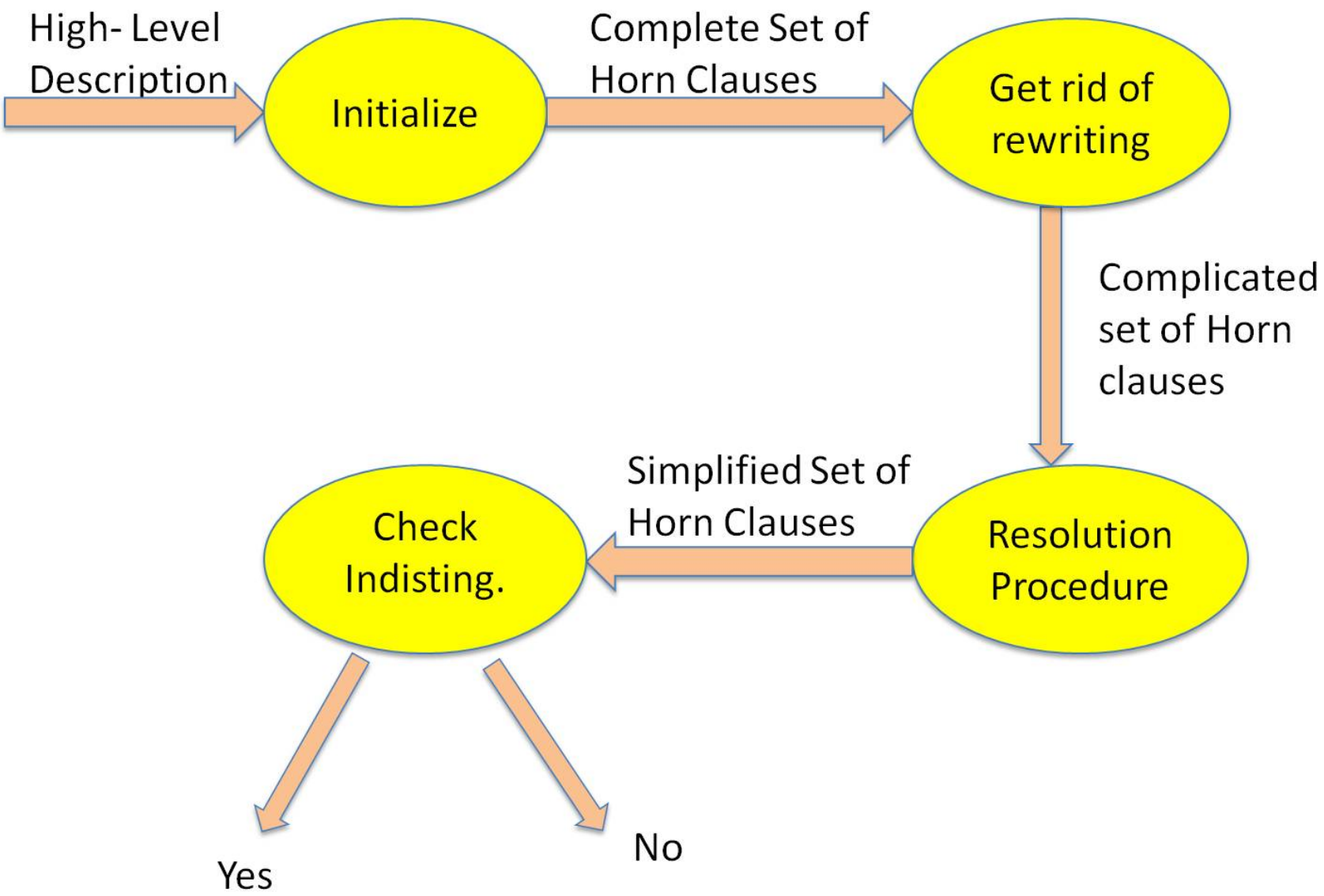
A trace is a determinate process

Definition:

P is *determinate* if whenever $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (T, \varphi)$ and $(P, \emptyset) \stackrel{\ell_1,\ldots,\ell_n}{\Longrightarrow} (T', \varphi')$ then $\varphi \approx_s \varphi'$.

Details of the procedure

Procedure in a nutshell



1. Horn clause modelling

$$T \quad = \quad \mathsf{send}(c, \mathsf{enc}(a, k)).\mathsf{receive}(c, x).[x \stackrel{?}{=} a].\mathsf{send}(c, s)$$

Compute a initial set for trace T: $\mathsf{seed}(T)$

$$\mathsf{k}(w_1, \mathsf{enc}(a, k))$$
$$\mathsf{k}(w_2, s) \Leftarrow \mathsf{k}(X, x), x =_{\mathsf{R}} a$$

$k(R, t)$: attacker knowledge predicate (*attacker can compute t using recipe R*)

1. Horn clause modelling

$$T \quad = \quad \mathsf{send}(c, \mathsf{enc}(a, k)).\mathsf{receive}(c, x).[x \stackrel{?}{=} a].\mathsf{send}(c, s)$$

Compute a initial set for trace T: $\mathsf{seed}(T)$

$$\mathsf{k}_{\mathsf{send}(c)}(w_1, \mathsf{enc}(a, k))$$
$$\mathsf{k}_{\mathsf{send}(c),\mathsf{receive}(c,x),\mathsf{send}(c)}(w_2, s) \Leftarrow \mathsf{k}_{\mathsf{send}(c)}(X, x), x =_{\mathsf{R}} a.$$

$k(R, t)$: attacker knowledge predicate (*attacker can compute t using recipe R*)

Add history for accuracy (avoid false attacks)

1. Horn clause modelling

$$T \quad = \quad \mathsf{send}(c, \mathsf{enc}(a, k)).\mathsf{receive}(c, x).[x \stackrel{?}{=} a].\mathsf{send}(c, s)$$

Compute a initial set for trace T: $\mathsf{seed}(T)$

$$\mathsf{k}_{\mathsf{send}(c)}(w_1, \mathsf{enc}(a, k))$$
$$\mathsf{k}_{\mathsf{send}(c),\mathsf{receive}(c,x),\mathsf{send}(c)}(w_2, s) \Leftarrow \mathsf{k}_{\mathsf{send}(c)}(X, x), x =_{\mathsf{R}} a.$$

$k(R, t)$: attacker knowledge predicate (*attacker can compute t using recipe R*)

Add history for accuracy (avoid false attacks)

Clauses for attacker capabilities:
$\mathsf{k}_w(f(X_1, \ldots X_n), f(x_1, \ldots, x_k)) \Leftarrow \mathsf{k}_w(X_1, x_1), \ldots, \mathsf{k}_w(X_k, x_k)$

1. Horn clause modelling: predicates

Predicates: interpreted over ground trace

- Reachability predicate $T \models \mathsf{r}_{\ell_1,\ldots,\ell_n}$ if $(T, \emptyset)$ can execute $\ell_1, \ldots \ell_n$
- attacker Knowledge predicate
  $T \models \mathsf{k}_{\ell_1,\ldots,\ell_n}(R, t)$ if whenever $(T, \emptyset)$ executes $\ell_1, \ldots \ell_n$
  then the attacker can derive t using recipe R
- Identity predicate
  $T \models \mathsf{i}_{\ell_1,\ldots,\ell_n}(R, R')$ if whenever $(T, \emptyset)$ executes $\ell_1, \ldots \ell_n$
  then the recipes R and R' yield the same term
- reachable identity predicates
  $T \models \mathsf{ri}_{\ell_1,\ldots,\ell_n}(R, R')$ if $T \models \mathsf{i}_{\ell_1,\ldots,\ell_n}(R, R')$ and $T \models \mathsf{r}_{\ell_1,\ldots,\ell_n}$

2. Getting rid of equations

Use equational unification to remove tests:

$$\left(H \Leftarrow B_1, \ldots, B_n, u =_{\mathsf{R}} v\right) \quad \rightsquigarrow \quad \begin{array}{l} \left((H \Leftarrow B_1, \ldots, B_n)\sigma_1\right) \\ \ldots \\ \left((H \Leftarrow B_1, \ldots, B_n)\sigma_k\right) \end{array}$$

where $\sigma_1, \ldots, \sigma_k$ is a complete set of unifiers for $u =_{\mathsf{R}} v$.

2. Getting rid of equations

Use equational unification to remove tests:

$$\Big(H \Leftarrow B_1, \ldots, B_n, u =_{\mathsf{R}} v\Big) \quad \rightsquigarrow \quad \begin{array}{l} \Big((H \Leftarrow B_1, \ldots, B_n)\sigma_1\Big) \\ \ldots \\ \Big((H \Leftarrow B_1, \ldots, B_n)\sigma_k\Big) \end{array}$$

where $\sigma_1, \ldots, \sigma_k$ is a complete set of unifiers for $u =_{\mathsf{R}} v$.

Use finite variant property to get rid of equational reasoning:

$$\Big(\mathsf{k}_w(R, t) \Leftarrow B_1, \ldots, B_n\Big) \quad \rightsquigarrow \quad \begin{array}{l} \Big((\mathsf{k}_W(R, t))\theta_1{\downarrow} \Leftarrow B_1\theta_1{\downarrow}, \ldots, B_n\theta_1{\downarrow}\Big) \\ \ldots \\ \Big((\mathsf{k}_W(R, t))\theta_k{\downarrow} \Leftarrow B_1\theta_k{\downarrow}, \ldots, B_n\theta_k{\downarrow}\Big). \end{array}$$

where $\theta_1, \ldots, \theta_k$ is a complete set of variants for t.

We can compute finite sets of variants and $\mathsf{mgu}_{\mathsf{E}}$ for the class of optimally reducing theories (contains subterm convergent, blind sigs, td commitment, ...)

Horn clause modelling: correctness

$\mathcal{H}(K)$: least Herbrand model of the set of Horn clauses K.

Theorem (Correctness of Horn clause modelling)

Let T be a ground trace.

- *(Soundness.) For any $f \in \mathsf{seed}(T) \cup \mathcal{H}(\mathsf{seed}(T))$ we have that $T \models f$.*
- *(Completeness.) If $(T, \emptyset) \xrightarrow{L_1,\ldots,L_m} (S, \varphi)$ then*
  1. $\mathsf{r}_{L_1\varphi\downarrow,\ldots,L_m\varphi\downarrow} \in \mathcal{H}(\mathsf{seed}(T))$, *and*
  2. *if $\varphi \vdash^R t$ then $\mathsf{k}_{L_1\varphi\downarrow,\ldots,L_m\varphi\downarrow}(R, t\downarrow) \in \mathcal{H}(\mathsf{seed}(T))$.*

3. Saturation: goals of saturation

Aims of saturation

- completeness of identity predicates
- completeness for solved clauses

  A clause is called solved if it is of the form
  $H \Leftarrow \mathsf{k}_{w_1}(X_1, x_1), \ldots, \mathsf{k}_{w_k}(X_k, x_k)$.

  For a set of solved clauses K checking $f \in \mathcal{H}(K)$ is easy
  (simple recursive algorithm)
  $\rightsquigarrow$ needed for checking indistinguishability

2. Saturation rules

Saturate seed knowledge base using the following rules

$$\text{Resolution}\ \frac{\begin{array}{c} f \in K, g \in K_{\text{solved}}, \qquad f = \Big(H \Leftarrow \mathsf{k}_{uv}(X,t), B_1, \ldots, B_n\Big) \\ g = \Big(\mathsf{k}_w(R,t') \Leftarrow B_{n+1}, \ldots, B_m\Big) \\ \sigma = \text{mgu}(\mathsf{k}_u(X,t), \mathsf{k}_w(R,t')) \qquad t \notin \mathcal{X} \end{array}}{K = K \oplus h \text{ where } h = \Big((H \Leftarrow B_1, \ldots, B_m)\sigma\Big)}$$

$$\text{Equation}\ \frac{\begin{array}{c} f, g \in K_{\text{solved}}, \qquad f = \Big(\mathsf{k}_u(R,t) \Leftarrow B_1, \ldots, B_n\Big) \\ g = \Big(\mathsf{k}_{u'v'}(R',t') \Leftarrow B_{n+1}, \ldots, B_m\Big) \\ \sigma = \text{mgu}(\mathsf{k}_u(\_,t), \mathsf{k}_{u'}(\_,t')) \end{array}}{K = K \oplus h \text{ where } h = \Big((\mathsf{i}_{u'v'}(R,R') \Leftarrow B_1, \ldots, B_m)\sigma\Big)}$$

$$\text{Test}\ \frac{\begin{array}{c} f, g \in K_{\text{solved}}, \qquad f = \Big(\mathsf{i}_u(R,R') \Leftarrow B_1, \ldots, B_n\Big) \\ g = \Big(\mathsf{r}_{u'v'} \Leftarrow B_{n+1}, \ldots, B_m\Big) \qquad \sigma = \text{mgu}(u,u') \end{array}}{K = K \oplus h \text{ where } h = \Big((\mathsf{ri}_{u'v'}(R,R') \Leftarrow B_1, \ldots, B_m)\sigma\Big)}$$

2. Saturation rules: soundness, completeness, termination

- The set of statements in $\mathcal{H}(\mathit{solved}(\mathit{sat}(\mathit{seed}(T))))$ is sound and complete

2. Saturation rules: soundness, completeness, termination

- The set of statements in $\mathcal{H}(solved(sat(seed(T))))$ is sound and complete
- Termination: failed to prove it :-(
  Conjectured for subterm convergent equational theories. Prototype implementation provides empirical evidence.

4. Checking indistinguishability

To check that $T \sqsubseteq_{ct} Q$

1. saturate: let $K = \mathsf{sat}(\mathsf{seed}(T))_{\mathsf{solved}}$
2. check reachability:
   for each $\mathsf{r}_{L_1,\ldots,L_n} \Leftarrow \mathsf{k}_{w_1}(X_1, x_1), \ldots \mathsf{k}_{w_k}(X_k, x_k) \in K$
   such that for all $1 \leq i \leq n$ $\mathsf{k}_{w_i}(X_i, x_i) \in \mathcal{H}(K)$
   check that $Q, \emptyset \xrightarrow{L_1,\ldots,L_n} Q', \varphi$
3. check static inclusion:
   for each $\mathsf{ri}_{L_1,\ldots,L_n}(R_1, R_2) \Leftarrow \mathsf{k}_{w_1}(X_1, x_1), \ldots \mathsf{k}_{w_k}(X_k, x_k) \in K$
   such that for all $1 \leq i \leq n$ $\mathsf{k}_{w_i}(X_i, x_i) \in \mathcal{H}(K)$
   check that $Q, \emptyset \xrightarrow{L_1,\ldots,L_n} Q', \varphi$ and $(R_1 = R_2)\varphi$

Perspectives

- Procedure for checking indistinguishability properties for bounded number of sessions
- Implementation:
  (anonymity in FOO and Okamoto electronic voting protocol, strong secrecy of NSPK, resistance to guessing attacks of EKE, ...)

Perspectives

- Procedure for checking indistinguishability properties for bounded number of sessions
- Implementation:
  (anonymity in FOO and Okamoto electronic voting protocol, strong secrecy of NSPK, resistance to guessing attacks of EKE, ...)

Work that still needs to be done

- More examples and evaluation of the tool
- Negative tests (else branches)
- Performance: optimize, parallelize, check bisimulation
  examples may take a few mins on a laptop (e.g. $\sim$2 mins for FOO)