Reducing search space for trace equivalence checking

FOSAD 2013

Lucca Hirschi

LSV, ENS Cachan

September 5, 2013

joint work with David Baelde (LSV) *and* Stéphanie Delaune (LSV)

Context

Prove automatically security properties of cryptographic protocols using formal methods.

Context

Prove automatically security properties of cryptographic protocols using formal methods.

Tools

- Applied-π models protocols (Dolev-Yao model);

Context

Prove automatically security properties of cryptographic protocols using formal methods.

Tools

- Applied-π models protocols (Dolev-Yao model);
- reachability or equivalence model security properties;

Context

Prove automatically security properties of cryptographic protocols using formal methods.

Tools

- Applied-π models protocols (Dolev-Yao model);
- reachability or equivalence model security properties;
- algorithms check reachability or equivalence.

Context

Prove automatically security properties of cryptographic protocols using formal methods.

Tools

- Applied-π models protocols (Dolev-Yao model);
- reachability or equivalence model security properties;
- algorithms check reachability or equivalence.

Issue

Main bottleneck: size of search space (interleavings).

Context

Prove automatically security properties of cryptographic protocols using formal methods.

Tools

- Applied-π models protocols (Dolev-Yao model);
- reachability or equivalence model security properties;
- algorithms check reachability or equivalence.

Issue

Main bottleneck: size of search space (interleavings).

Our Contribution

Reduce search space of equivalence checking using POR ideas by eliminating a lot of redundancies.

Our Contribution

Reduce search space of equivalence checking using POR ideas by eliminating a lot of redundancies.

Sebastian Mödersheim, Luca Vigano, and David Basin.
Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols.
Journal of Computer Security, 18(4):575–618, 2010.

Outline

Outline

Applied-π

Terms

$\mathcal{T}$: a given set of terms modulo an equational theory. E.g. $\mathrm{dec}(\mathrm{enc}(m, k), k) = m$.

Simple Processes

- $P_c ::= 0 \mid [T]in(c, x) \mid [T]out(c, m).P_c \qquad m \in \mathcal{T}$
- $P_s ::= P_{c_1} | P_{c_2} | \dots P_{c_n} \qquad c_i \neq c_j$

Applied-π

Terms

$\mathcal{T}$: a given set of terms modulo an equational theory. E.g. $\mathrm{dec}(\mathrm{enc}(m, k), k) = m$.

Simple Processes

- $P_c ::= 0 \mid [T]in(c, x) \mid [T]out(c, m).P_c \qquad m \in \mathcal{T}$
- $P_s ::= P_{c_1} | P_{c_2} | \ldots P_{c_n} \qquad c_i \neq c_j$
- Process: $(P_s;\ \Phi)$ (Φ set of messages revealed to the intruder).

Applied-π

Terms

$\mathcal{T}$: a given set of terms modulo an equational theory. E.g. $\mathrm{dec}(\mathrm{enc}(m,k),k) = m$.

Simple Processes

- $P_c ::= 0 \mid [T]in(c,x) \mid [T]out(c,m).P_c \qquad m \in \mathcal{T}$
- $P_s ::= P_{c_1} | P_{c_2} | \dots P_{c_n} \qquad c_i \neq c_j$
- Process: $(P_s;\ \Phi)$ (Φ set of messages revealed to the intruder).

Semantics

$$(\{[T].out(c,m).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\nu w.out(c,w)} (\{P\} \uplus \mathcal{P}; \Phi \cup \{w \rhd m\})$$
$$\text{if } T \ \wedge \ w \text{ fresh in } \Phi$$

$$(\{in(c,x).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{in(c,t)} (\{P[x \mapsto u]\} \cup \mathcal{P}; \Phi)$$
$$\text{if } t\Phi = u \ \wedge \ \mathrm{fv}(t) \subseteq \mathrm{dom}(\Phi)$$

Equivalence

Trace equivalence

- $\Phi \sim \Phi' \iff \forall M, N,\ M\Phi = N\Phi \iff M\Phi' = N\Phi'$ and conversely;
- $A \approx B \iff \forall A \xrightarrow{s} A',\ \exists B',\ B \xrightarrow{s} B' \wedge \Phi_{A'} \sim \Phi_{B'}$ and conversely.

Trace equivalence allows to model anonymity, unlikability, etc.

Equivalence

Trace equivalence

- $\Phi \sim \Phi' \iff \forall M, N,\ M\Phi = N\Phi \iff M\Phi' = N\Phi'$ and conversely;
- $A \approx B \iff \forall A \xrightarrow{s} A',\ \exists B',\ B \xrightarrow{s} B' \wedge \Phi_{A'} \sim \Phi_{B'}$ and conversely.

Trace equivalence allows to model anonymity, unlikability, etc.

Our aim

Improve algorithms/programs checking trace equivalence (for simple processes).

Symbolic calculus - 1

Inputs messages: infinitely branching $\rightsquigarrow$ symbolic calculus.

Symbolic calculus - 1

Inputs messages: infinitely branching $\rightsquigarrow$ symbolic calculus.

System of Constraints

- Constraints: $(X \rhd x)$; $u = v$, $(\mathrm{fv}^?(X) : \mathrm{dom}(\Phi))$;
- System of constraints: $(\Phi, \mathcal{D})$.

Symbolic calculus - 1

Inputs messages: infinitely branching $\rightsquigarrow$ symbolic calculus.

System of Constraints

- Constraints: $(X \triangleright x)$; $u = v$, $(\mathrm{fv}^?(X) : \mathrm{dom}(\Phi))$;
- System of constraints: $(\Phi, \mathcal{D})$.

$$P = out(c,k).in(c,x).out(c,\langle k,x\rangle).in(c,y)$$

leads to

$$\mathcal{D} = \{X \triangleright x; Y \triangleright y; (\mathrm{fv}^?(X) : \{w\}); (\mathrm{fv}^?(Y) = \{w; w'\})\}$$
$$\Phi = \{w \triangleright k; w' \triangleright \langle k, x\rangle\}$$

Symbolic calculus - 1

Inputs messages: infinitely branching $\rightsquigarrow$ symbolic calculus.

System of Constraints

- Constraints: $(X \rhd x)$; $u = v$, $(\mathrm{fv}^?(X) : \mathrm{dom}(\Phi))$;
- System of constraints: $(\Phi, \mathcal{D})$.

$$P = out(c,k).in(c,x).out(c,\langle k,x\rangle).in(c,y)$$

leads to

$$\mathcal{D} = \{X \rhd x; Y \rhd y; (\mathrm{fv}^?(X) : \{w\}); (\mathrm{fv}^?(Y) = \{w; w'\})\}$$
$$\Phi = \{w \rhd k; w' \rhd \langle k,x\rangle\}$$

Symbolic processes

$$(\mathcal{P}; \Phi; \mathcal{D}; tr)$$

Symbolic Calculus - 2

Semantics:

$$(\{[T].out(c,m).P\} \uplus \mathcal{P}; \Phi; \mathcal{D}; tr) \xrightarrow{\nu w.out(c,X)}_s (\{P\} \uplus \mathcal{P}; \Phi \cup \{w \rhd m\}; \mathcal{D} \cup \{T\}; tr.\nu w.out(c,X))$$

if w fresh in ϕ

$$(\{[T].in(c,x).P\} \uplus \mathcal{P}; \Phi; \mathcal{D}; tr) \xrightarrow{in(c,X)}_s (\mathcal{P}; \Phi; \mathcal{D} \cup \{T; (X \rhd x); (\mathrm{fv}^?(X) : \mathrm{dom}(\Phi))\}; tr.in(c,X))$$

Symbolic Calculus - 2

Semantics:

$$(\{[T].out(c,m).P\} \uplus \mathcal{P}; \Phi; \mathcal{D}; tr) \xrightarrow{\nu w.out(c,X)}_s (\{P\} \uplus \mathcal{P}; \Phi \cup \{w \rhd m\}; \mathcal{D} \cup \{T\}; tr.\nu w.out(c,X))$$

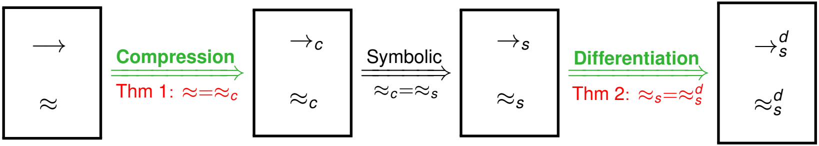
if w fresh in ϕ

$$(\{[T].in(c,x).P\} \uplus \mathcal{P}; \Phi; \mathcal{D}; tr) \xrightarrow{in(c,X)}_s (\mathcal{P}; \Phi; \mathcal{D} \cup \{T; (X \rhd x); (\mathrm{fv}^?(X) : \mathrm{dom}(\Phi))\}; tr.in(c,X))$$

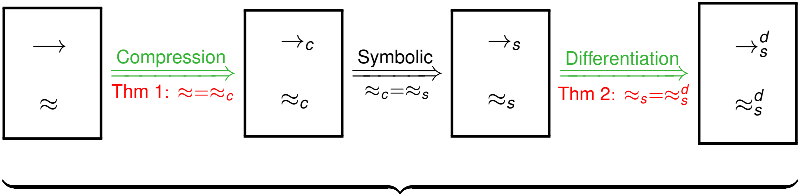
Symbolic equivalence

$A \approx_s B \iff \forall A \xrightarrow{s}_s A' \; \forall \Theta \in \mathcal{S}\mathrm{ol}(\Phi_{A'}, \mathcal{D}_{A'}),\ \exists B' \; B \xrightarrow{s}_s B', \Theta \in \mathcal{S}\mathrm{ol}(\Phi_{B'}, \mathcal{D}_{B'})$ and $\Phi_{A'} \sim \Phi_{B'}$ and conversely.

Outline

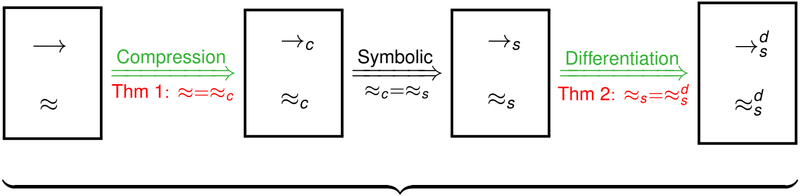


$\longrightarrow$
$\approx$
Compression
Thm 1: $\approx = \approx_c$
$\longrightarrow_c$
$\approx_c$
Symbolic
$\approx_c = \approx_s$
$\longrightarrow_s$
$\approx_s$
Differentiation
Thm 2: $\approx_s = \approx_s^d$
$\longrightarrow_s^d$
$\approx_s^d$



Apply optimizations to SPEC:

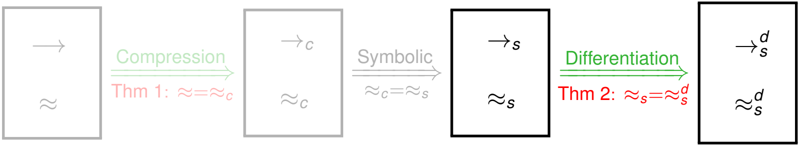
- adpat its formalism;
- constraints solving.



Apply optimizations to SPEC:

- adpat its formalism;
- constraints solving.

Implementation

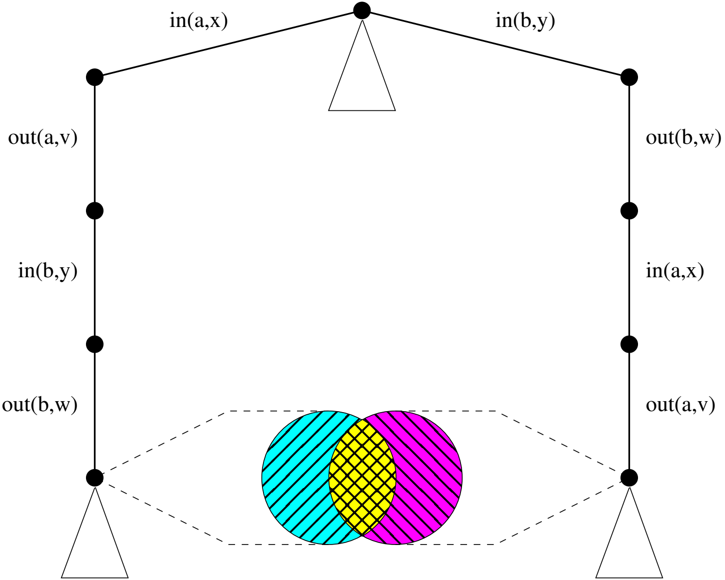


Apply optimizations to SPEC:

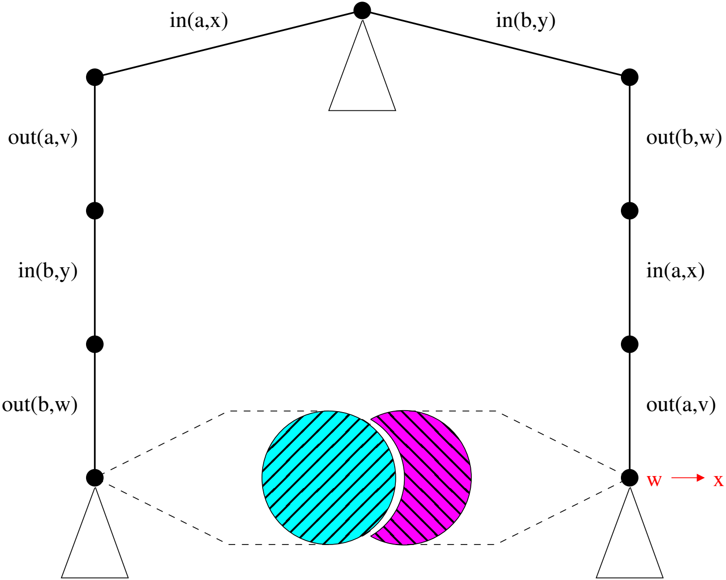
- adpat its formalism;
- constraint reduction.

Implementation

Outline

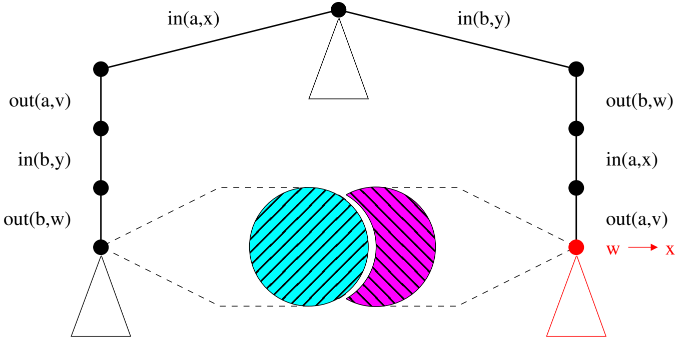


$$P = in(a,x).out(a,k).P_a \mid in(b,y).out(b,k').P_b$$



$$P = in(a,x).out(a,k).P_a \mid in(b,y).out(b,k').P_b$$

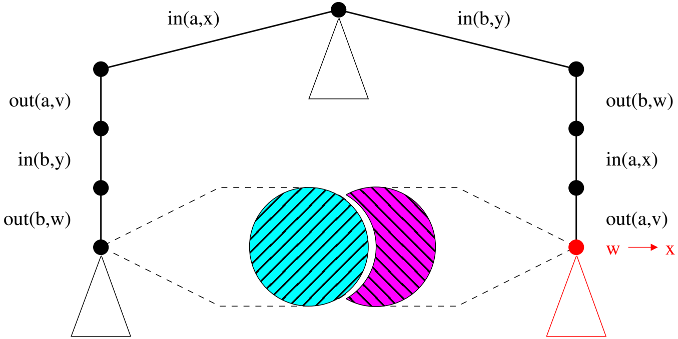
Dependency constraints



Dependency constraint: $w \in$ message of x

We can add constraints on the fly.

Dependency constraints

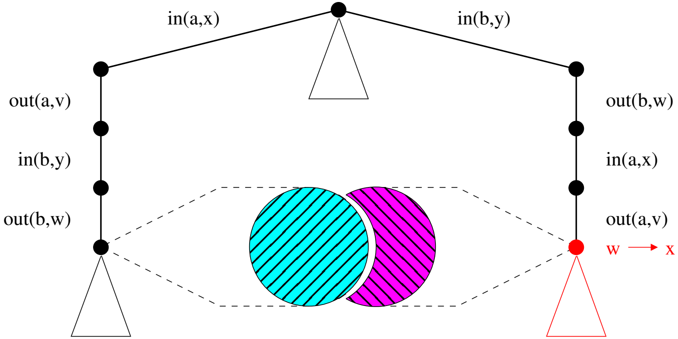


Dependency constraint: $w \in$ message of x

We can add constraints on the fly.

- Eliminate symmetric traces;

Dependency constraints

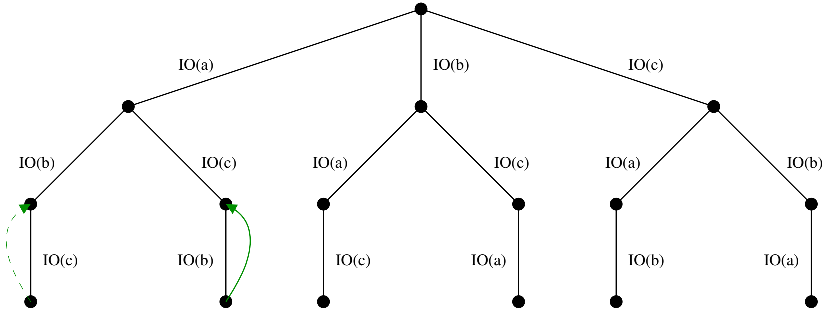


Dependency constraint: $w \in$ message of x

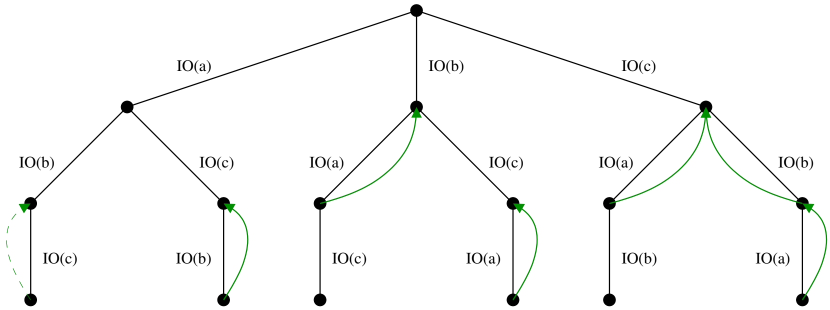
We can add constraints on the fly.

- Eliminate symmetric traces;
- Do not remove too much information (intruder can observe the order).

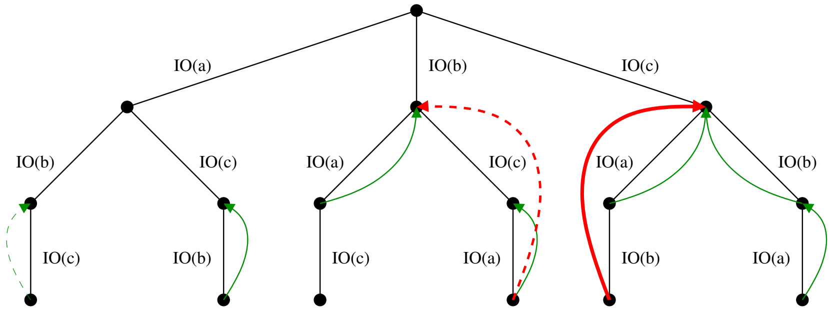
$$P = IO(a)|IO(b)|IO(c) \text{ where } IO(x) = in(x, X).out(x, w_x)$$



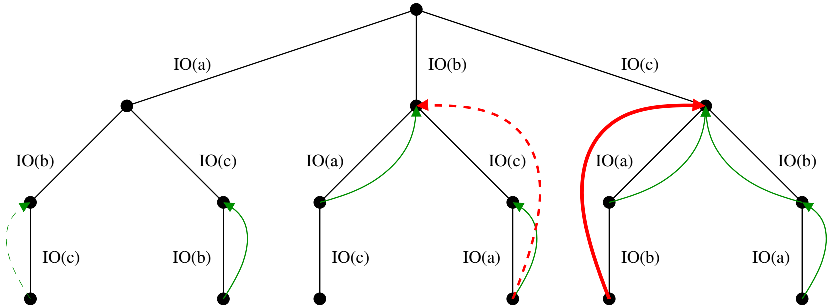
$$P = IO(a)|IO(b)|IO(c) \text{ where } IO(x) = in(x, X).out(x, w_x)$$



$$P = IO(a)|IO(b)|IO(c) \text{ where } IO(x) = in(x, X).out(x, w_x)$$



$$P = IO(a)|IO(b)|IO(c) \text{ where } IO(x) = in(x, X).out(x, w_x)$$



- $c_n < c_1$; $\xrightarrow{t=IO(c_1).IO(c_2)\ldots IO(c_n)} s \rightsquigarrow \xrightarrow{IO(c_n).IO(c_1)\ldots IO(c_{n-1})} s$
- $c_2, c_3 \ldots c_{n-1} < c_n$

$\mathcal{G}(t)$ = there exists $1 \leq i < n$ such that $w_i \in$ message of x_n

Differentiation

Differentiated semantics

Symbolic semantics + dependency constraints built on the fly.

Differentiation

Differentiated semantics

Symbolic semantics + dependency constraints built on the fly.

$$(\{in(c,x).out(c,m).P\} \uplus \mathcal{P}; \mathcal{D}; \Phi; t) \xrightarrow{io(c,X,w)}{}^{d}_{s}$$
$$(\{P\} \uplus \mathcal{P}; \mathcal{D} \cup \{(X \rhd x), \mathcal{G}(t.io(c,X,w))\}; \Phi \cup \{w \rhd m\}; t.io(c,X,w))$$

$\rightsquigarrow$ less solutions, less traces/interleavings to check.

Differentiation

Differentiated semantics

Symbolic semantics + dependency constraints built on the fly.

$$(\{in(c,x).out(c,m).P\} \uplus \mathcal{P}; \mathcal{D}; \Phi; t) \xrightarrow{io(c,X,w)}{}^{d}_{s}$$
$$(\{P\} \uplus \mathcal{P}; \mathcal{D} \cup \{(X \rhd x), \mathcal{G}(t.io(c,X,w))\}; \Phi \cup \{w \rhd m\}; t.io(c,X,w))$$

$\rightsquigarrow$ less solutions, less traces/interleavings to check.

Theorem

$$\approx^{d}_{s} = \approx_{s}$$

Idea of the proof

- $[t]$: set of traces modulo valid permutations;
- $\mathrm{Min}([t])$: lexico. minimum of the class.

Lemma 1

If P has an trace t then it has all traces of $[t]$.

Lemma 2

- If P has an trace t then it has a differentiated trace $\mathrm{Min}(\mathrm{t})$;
- P has no other differentiated trace in $[t]$.

Outline

Conclusion

- Better differentiation (compression, semantics, extended patterns) for simple processes;
- applied to trace equivalence checking.
- implementation in SPEC.

Conclusion

- Better differentiation (compression, semantics, extended patterns) for simple processes;
- applied to trace equivalence checking.
- implementation in SPEC.

Protocol	# ac .	T. REF (s)	T. OPT (s)
3 parallels	8	44.59	5.88
7 parallels	16	∞	370.65
depth 4	10	42.87	8.42
depth 10	22	∞	122.27
WMF, auth. false, 1 sess.	12	30.89	1.87
WMF, auth., 1 sess.	14	51.54	6.43
WMF, strong secr., 1 sess.	16	65.20	8.09
WMF, false, 2 sess.	24	7742.24	3.30
NSSK, auth., 1 session	10	76.68	22.99
Yahalom, auth., 1 session	10	6602.82	237.10

Conclusion

- Better differentiation (compression, semantics, extended patterns) for simple processes;
- applied to trace equivalence checking.
- implementation in SPEC.

Protocol	# ac .	T. REF (s)	T. OPT (s)
3 parallels	8	44.59	5.88
7 parallels	16	∞	370.65
depth 4	10	42.87	8.42
depth 10	22	∞	122.27
WMF, auth. false, 1 sess.	12	30.89	1.87
WMF, auth., 1 sess.	14	51.54	6.43
WMF, strong secr., 1 sess.	16	65.20	8.09
WMF, false, 2 sess.	24	7742.24	3.30
NSSK, auth., 1 session	10	76.68	22.99
Yahalom, auth., 1 session	10	6602.82	237.10

Future Work

- Richer class of processes;
- improve constraints solving.