Using Interpolation for the Verification of Security Protocols

Marco Rocchetto, **Luca Viganò**, Marco Volpe

REG**IS**: **RE**search **G**roup in **I**nformation **S**ecurity
Department of Computer Science
Università di Verona, Italy

Department of Informatics
King's College London, UK

iPRA, July 18, 2014

Outline

1. The idea
2. SPiM
   - Method description
   - Example
3. SPiM Java prototype
4. Future work

The idea behind SPiM

research group in information security

Interpolation

- Successfully applied in formal methods for model checking and test-case generation for sequential programs

Security protocols

- Unsuitable to the direct application of such methods:
  - sequential programs only
  - no intruder logic

The idea behind SPiM

REGIS research group in information security

Interpolation

- Successfully applied in formal methods for model checking and test-case generation for sequential programs

Security protocols

- Unsuitable to the direct application of such methods:
  - sequential programs only
  - no intruder logic

SPiM (Security Protocol interpolation Method)

- Given a formal protocol specification, it combines
  - Craig interpolation,
  - symbolic execution,
  - standard Dolev-Yao intruder model

  to search for goals (i.e., possible attacks on the protocol)
- **Interpolants**: generated as a response to search failure in order to prune possible useless traces and speed up exploration

Outline

REGIS research group in information security

1 The idea

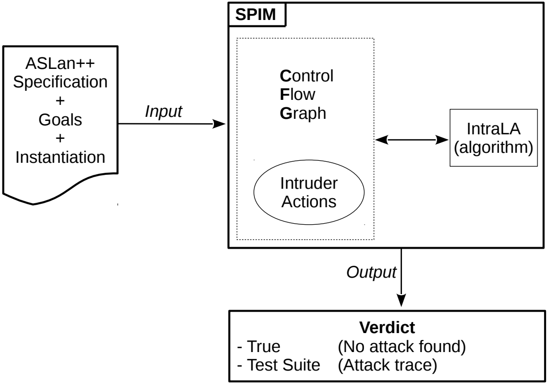
2 SPiM
- Method description
- Example

3 SPiM Java prototype

4 Future work

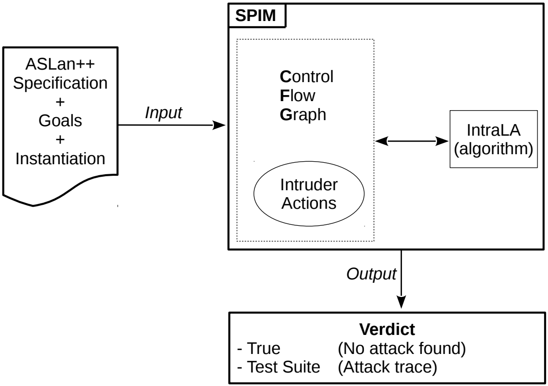
General overview of SPiM

REGIS research group in information security



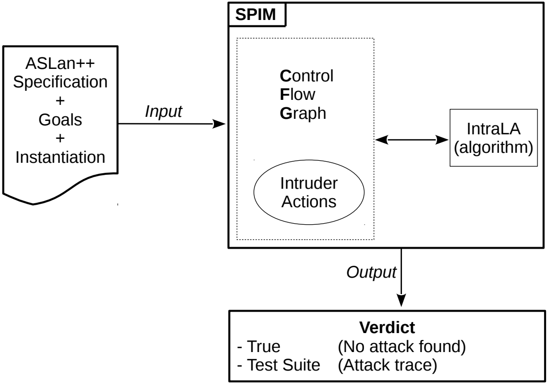
Starts from a specification of security protocol and property, and a scenario

General overview of SPiM



Creates and symbolically executes a sequential program (**control flow graph**) searching for set of goals (i.e., attacks)

General overview of SPiM

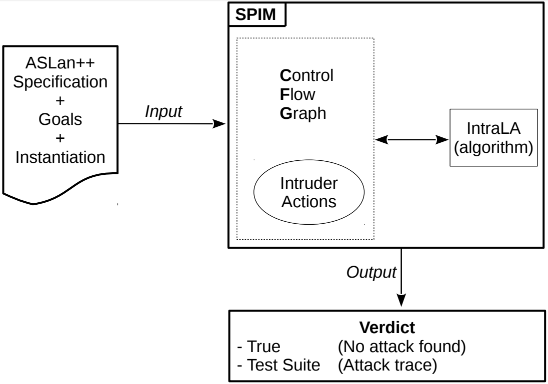


When a goal is reached

extracts attack trace (test case) from set of constraints produced in execution path

General overview of SPiM

REGIS research group in information security



When search fails to reach a goal

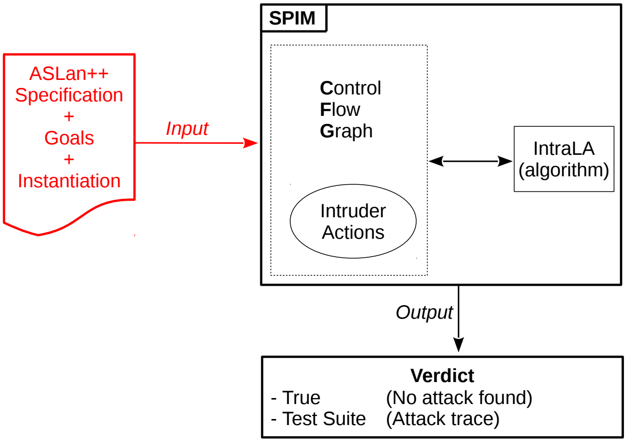
starts backtrack phase, during which nodes of graph are annotated (à la McMillan) with formulas obtained by using Craig interpolation

Interpolants: generated as a response to search failure in order to prune possible useless traces and speed up exploration

Outline

1. The idea
2. SPiM
   - Method description
   - Example
3. SPiM Java prototype
4. Future work

Input


ASLan++
Specification
+
Goals
+
Instantiation
Input
SPIM
Control
Flow
Graph
Intruder
Actions
IntraLA
(algorithm)
Output
Verdict
- True (No attack found)
- Test Suite (Attack trace)

An example

REGIS research group in information security

Needham-Schroeder Public Key (NSPK) protocol

$$A \rightarrow B \quad : \quad \{N_A, A\}_{pk(B)}$$
$$B \rightarrow A \quad : \quad \{N_A, N_B\}_{pk(A)}$$
$$A \rightarrow B \quad : \quad \{N_B\}_{pk(B)}$$

Man-in-the-middle attack

$$A \rightarrow i \quad : \quad \{N_A, A\}_{pk(i)}$$
$$i(A) \rightarrow B \quad : \quad \{N_A, A\}_{pk(B)}$$
$$B \rightarrow i(A) \quad : \quad \{N_A, N_B\}_{pk(A)}$$
$$i \rightarrow A \quad : \quad \{N_A, N_B\}_{pk(A)}$$
$$A \rightarrow i \quad : \quad \{N_B\}_{pk(i)}$$
$$i(A) \rightarrow B \quad : \quad \{N_B\}_{pk(B)}$$

An example

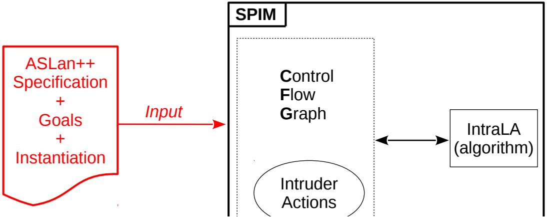
Needham-Schroeder with Lowe's fix (NSL) protocol

$$A \rightarrow B \quad : \quad \{N_A, A\}_{pk(B)}$$
$$B \rightarrow A \quad : \quad \{N_A, N_B, B\}_{pk(A)}$$
$$A \rightarrow B \quad : \quad \{N_B\}_{pk(B)}$$

Man-in-the-middle attack

Attack does not work anymore (other attacks do).

Input



ASLan++ NSL Code example

```
Alice(Actor,B:agent){

 Na:=fresh();
 Actor->B:{Actor.Na}_pk(B);
 B->Actor:{Na.?Nb.B}_pk(Actor);
 Actor->B:{Nb}_pk(B);

}
```

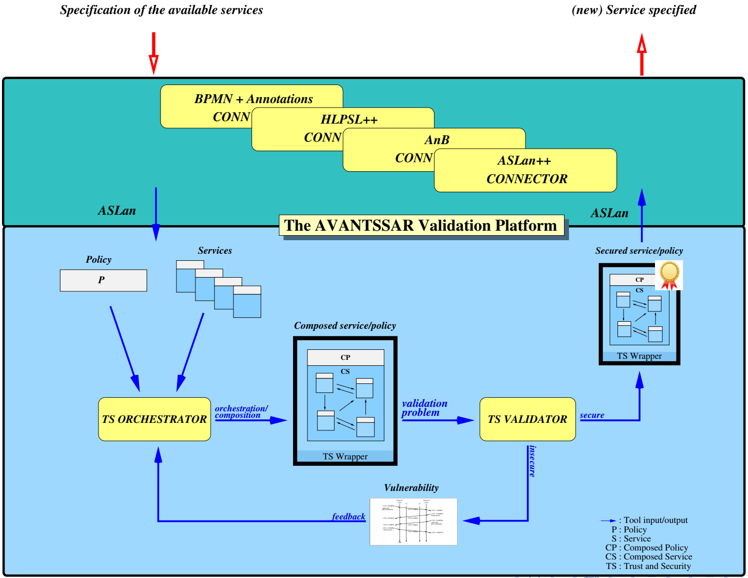
```
Bob(Actor,A:agent){

 ?->Actor:{?A.?Na}_pk(Actor);
 Nb:=fresh();
 Actor->A:{Na.Nb.B}_pk(A);
 A->Actor:{Nb}_pk(Actor);

}
```

Goal:
Bob authenticates Alice

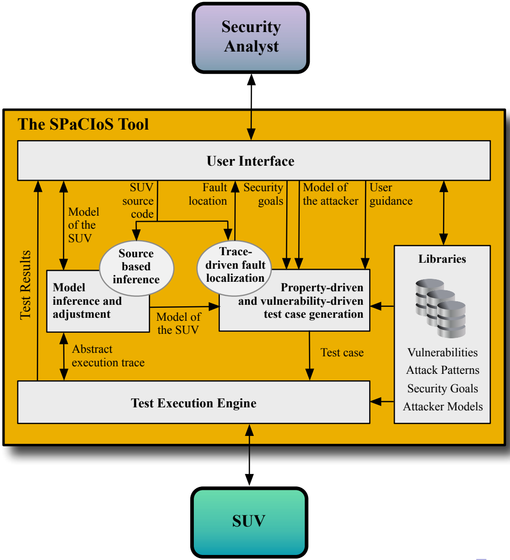
Instantiation:

	Alice	Bob
(1)	a	i
(2)	i	b

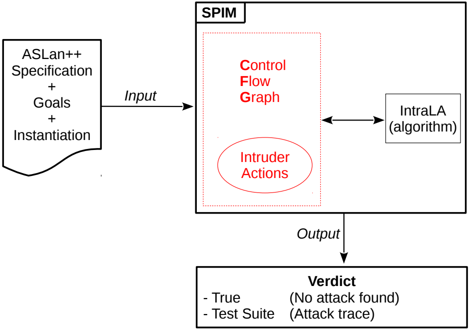
The AVANTSSAR Platform



The SPaCIoS Tool



Control Flow Graph and Intruder Actions



- IntraLA algorithm designed for sequential programs

  (K. McMillan. *Lazy annotation for program testing and verification*. CAV'10)

- To apply (a modified version of) IntraLA to security protocols, we define a translation of a specification of a protocol P for a given scenario into a sequential non-deterministic program

From parallel to sequential

$Alice := a, Bob := i$

```
1.1)  Alice.Actor := a;
1.2)  Alice.B := Y_1;
1.3)  IK := {a,b,i,pk_a,pk_b,pk_i,pk_i^-1};
1.4)
1.5)  Alice.Na := c_1;
1.6)  IK := IK + {Alice.Na,Alice.Actor}_pk(Alice.B);
1.7)
1.8)  if (IK |- {Alice.Na,?Alice.Nb,Alice.B}_pk(Alice.Actor))
1.9)    then
1.10)     Alice.Nb = Y_2;
1.11)    else
1.12)     end
1.13)
1.14)  IK := IK + {Alice.Nb}_pk(Alice.B);
```

```
Alice(Actor,B:agent){

 Na:=fresh();
 Actor->B:{Actor.Na}_pk(B);
 B->Actor:{Na.?Nb.B}_pk(Actor);
 Actor->B:{Nb}_pk(B);

}

Bob(Actor,A:agent){

 ?->Actor:{?A.?Na}_pk(Actor);
 Nb:=fresh();
 Actor->A:{Na.Nb.B}_pk(A);
 A->Actor:{Nb}_pk(Actor);

}
```

From parallel to sequential

REGIS research group in information security

$Alice := i, Bob := b$

```
Bob.Actor := b;
IK := {a,b,i,pk_a,pk_b,pk_i,pk_i^-1};

if (IK |- {?Bob.Na,?Bob.A}_pk(Bob.Actor))
  then
    Bob.Na = Y_1;
    Bob.A = Y_2;
  else
    end

Bob.Nb := c_1;
IK := IK + {Bob.Na,Bob.Nb,Bob.Actor}_pk(Bob.A);

if (IK |- {Bob.Nb}_pk(Bob.Actor))
  then
    do nothing
  else
    end
```

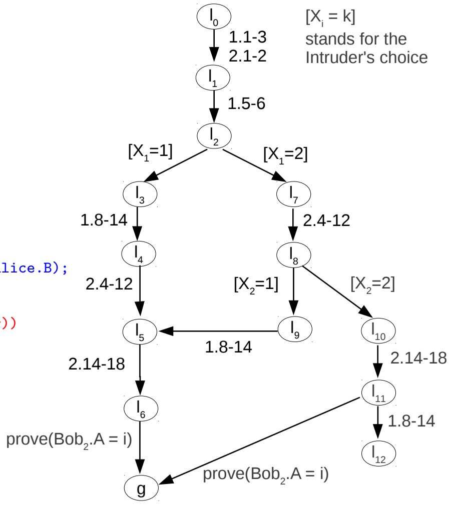
```
Alice(Actor,B:agent){

 Na:=fresh();
 Actor->B:{Actor.Na}_pk(B);
 B->Actor:{Na.?Nb.B}_pk(Actor);
 Actor->B:{Nb}_pk(B);

}

Bob(Actor,A:agent){

 ?->Actor:{?A.?Na}_pk(Actor);
 Nb:=fresh();
 Actor->A:{Na.Nb.B}_pk(A);
 A->Actor:{Nb}_pk(Actor);

}
```

Control Flow Graph - NSL

REGIS research group in information security

- Combining sessions
- Input variables X_i to switch between sessions

```
1.1)  Alice.Actor := a;
1.2)  Alice.B := Y_1;
1.3)  IK := {a,b,i,pk_a,pk_b,pk_i,pk_i^-1};
1.4)
1.5)  Alice.Na := c_1;
1.6)  IK := IK + {Alice.Na,Alice.Actor}_pk(Alice.B);
1.7)
1.8)  if (IK |-
 {Alice.Na,?Alice.Nb,Alice.B}_pk(Alice.Actor))
1.9)    then
1.10)      Alice.Nb = Y_2;
1.11)     else
1.12)      end
1.13)
1.14)  IK := IK + {Alice.Nb}_pk(Alice.B);
```



Correctness of the translation

Summing up

- Each ASLan++ statement is translated into a fragment of code in a pseudo programming language (SiL).
- The session interleaving is simulated in SiL by using conditionals with respect to input parameters.
- A (quite standard) notion of equivalence between ASLan++ and SiL states is defined.

Equivalence Theorem

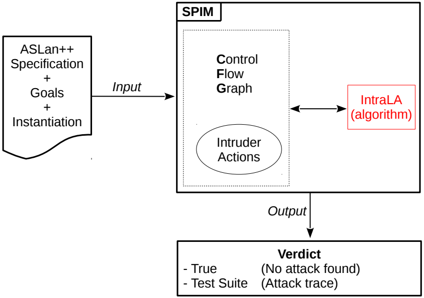
The original ASLan++ specification and its translation into SiL are "**equivalent**".

Proof By standard bisimulation techniques. In particular, we show that for each sequence of steps in ASLan++, there exists a path in the control flow graph of its SiL translation that passes through equivalent states.

Corollary

An **attack state** is found in an ASLan++ specification iff a **goal location** is reachable in its SiL translation.

IntraLA algorithm



Modified IntraLA algorithm executes symbolically a program graph searching for goal locations (attacks)

- If we fail to reach a goal, an **annotation** (condition under which no goal can be reached) produced by Craig interpolation
- Annotation (backtrack) propagated to other nodes to block a later phase of symbolic execution along an uninteresting run (that will not reach goal)

IntraLA [McMillan, CAV2010]

REGIS research group in information security

Init $\dfrac{}{\{l_0, s_0\},\ A_0,\ G_0}$

Decide $\dfrac{Q,\ A,\ G}{Q + (l_2, s_2),\ A,\ G}$

$$q = (l_1, s_1) \in Q$$
$$e = (l_1, a, l_2) \in \Delta$$
$$\neg B(q, A(e))$$
$$s_2 \in SI(a)(s_1)$$
$$\neg B((l_2, s_2), A(l_2))$$

Learn $\dfrac{Q,\ A,\ G}{Q,\ A + e\colon \phi,\ G}$

$$q = (l_1, s_1) \in Q$$
$$e = (l_1, a, l_2) \in \Delta$$
$$B(q, \phi)$$
$$J(e : \phi, A)$$

Conjoin $\dfrac{Q,\ A,\ G}{Q - q,\ A + l\colon \phi,\ G - l}$

$$q = (l, s) \in Q$$
$$\neg B(q, A(l))$$
$$\forall e \in Out(l), e :\ \phi_e \in A \wedge B(q, \phi_e)$$
$$\phi = \wedge\{\phi_e \mid e \in Out(l)\}$$

- Decide: symbolically executes one program action and generates a new **query** (keeps track of which symbolic states still need to be considered) from an existing one
- Learn used to generate annotations in backtrack phase
- Conjoin used to backtrack and merge annotations coming from different branches

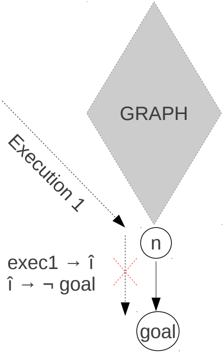
Interpolants as annotations

Craig's Interpolation

In FOL, if $\alpha \wedge \beta$ is inconsistent, then there exists $\hat{\imath}$ s.t.

- α implies $\hat{\imath}$
- $\hat{\imath}$ implies $\neg\beta$
- $\mathcal{L}(\hat{\imath}) \in \mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$

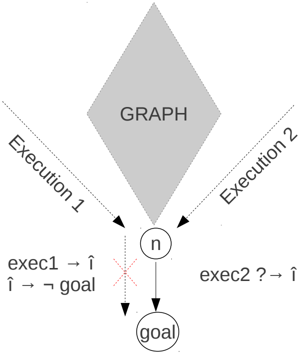
Interpolants as annotations



An example

After an "unsuccessful" execution 1, we calculate an interpolant î as a condition that prevents us to reach the goal, and **annotate** the location *n* with it.

Interpolants as annotations



An example

If execution 2 reaches in the same location a state where î is implied, then we can **ignore** that path (as we know that no goal will ever be reached).

Interpolants as annotations

REGIS research group in information security

Craig's Interpolation

In FOL, if $\alpha \wedge \beta$ is inconsistent, then there exists $\hat{\imath}$ s.t.

- α implies $\hat{\imath}$
- $\hat{\imath}$ implies $\neg\beta$
- $\mathcal{L}(\hat{\imath}) \in \mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$

We can define α and β as follows:

- $\alpha = PC \bigwedge_{v \in Var} v = Env(v)$
- $\beta = Sem(a) \wedge \neg ann'$

where:

- *PC* is a conjunction of *path constraints*
- *Var* is the set of *program variables*
- *Env* is the *environment*
- *Sem*(*a*) is the *semantics* (expressed as a transition formula) of the last *action a*
- *ann* is the current *annotation* of the node

Outline

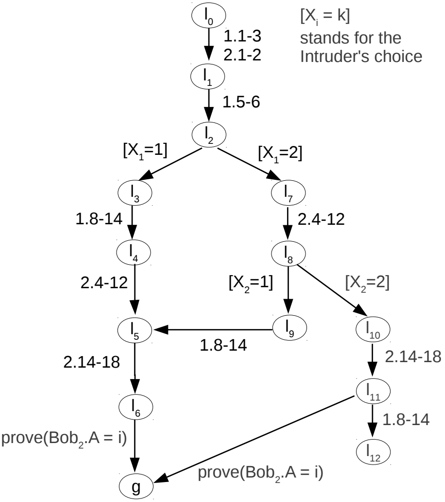
1 The idea

2 SPiM
- Method description
- Example

3 SPiM Java prototype

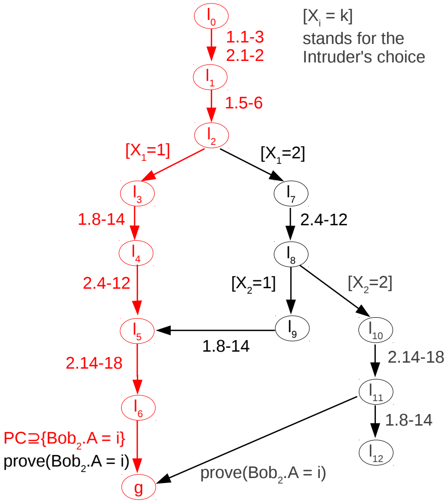
4 Future work

NSL example



[X_i = k] stands for the Intruder's choice

NSL example



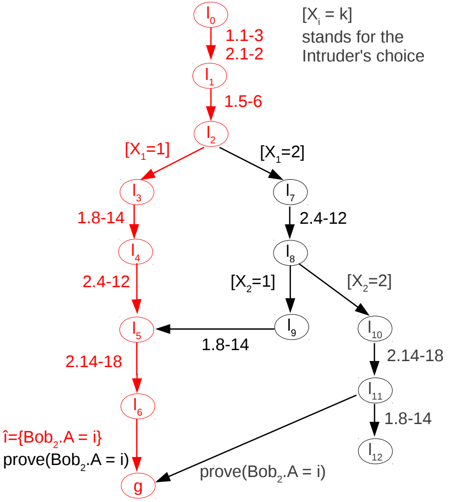
[X_i = k] stands for the Intruder's choice

NSL example

Learn on (I_6, g)

$\alpha \Rightarrow \hat{\imath} \Rightarrow \neg\beta$

$\hat{\imath} = \{Bob_2.A = i\}$



NSL example

Learn on (I_5, I_6)

$\alpha \Rightarrow \hat{\imath} \Rightarrow \neg\beta$

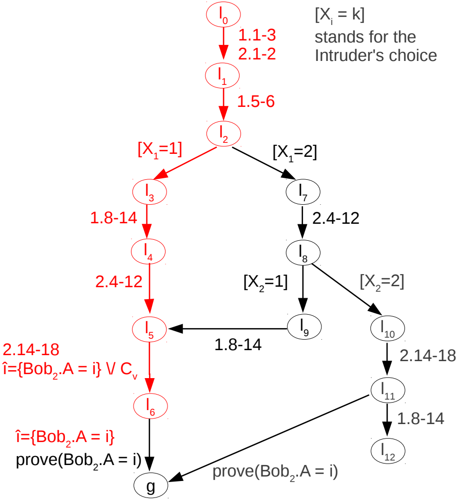
$\hat{\imath} = \{Bob_2.A = i\} \vee C_V$

$C_V \in \mathcal{L}(Var)$

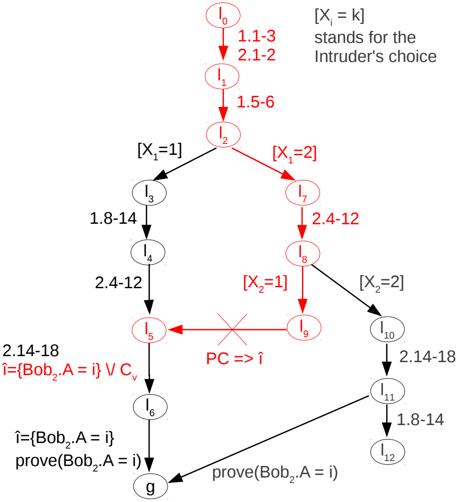
is a constraint

over Var s.t. C_V entails

$IK \nvdash \{Bob_2.Nb\}_{pk\{Bob2.Actor\}}$

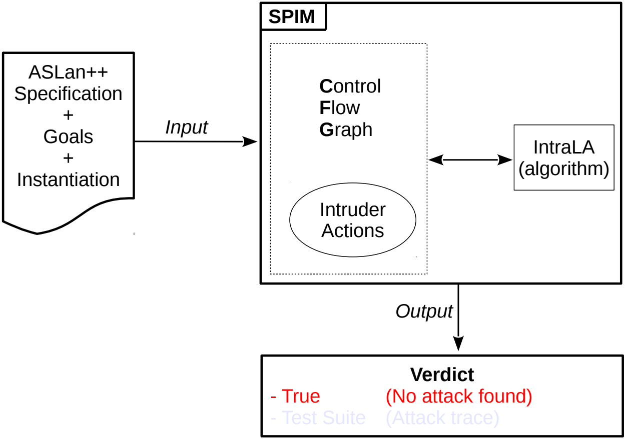


NSL example



[X_i = k]
stands for the
Intruder's choice

Verdict NSL


ASLan++
Specification
+
Goals
+
Instantiation
Input
SPIM
Control
Flow
Graph
Intruder
Actions
IntraLA
(algorithm)
Output
Verdict
- True
(No attack found)
- Test Suite
(Attack trace)

Verdict NSPK

Without Lowe's fix we obtain a MITM attack:

$$
\begin{array}{rcl}
Alice_1.Actor \rightarrow Alice_1.B & : & \{Alice_1.Na, Alice_1.Actor\}_{pk(Alice_1.B)} \\
? \rightarrow Bob_2.Actor & : & \{Bob_2.Na, Bob_2.A\}_{pk(Bob_2.Actor)} \\
Bob_2.Actor \rightarrow Bob_2.A & : & \{Bob_2.Na, Bob_2.Nb\}_{pk(Bob_2.A)} \\
Alice_1.B \rightarrow Alice_1.Actor & : & \{Alice_1.Na, Alice_1.Nb\}_{pk(Alice_1.Actor)} \\
Alice_1.Actor \rightarrow Alice_1.B & : & \{Alice_1.Nb\}_{pk(Alice_1.B)} \\
Bob_2.A \rightarrow Bob_2.Actor & : & \{Bob_2.Nb\}_{pk(Bob_2.Actor)}
\end{array}
$$

Verdict NSPK

Without Lowe's fix we obtain a MITM attack:

$$
\begin{array}{rcl}
Alice_1.Actor \rightarrow Alice_1.B & : & \{Alice_1.Na, Alice_1.Actor\}_{pk(Alice_1.B)} \\
? \rightarrow Bob_2.Actor & : & \{Bob_2.Na, Bob_2.A\}_{pk(Bob_2.Actor)} \\
Bob_2.Actor \rightarrow Bob_2.A & : & \{Bob_2.Na, Bob_2.Nb\}_{pk(Bob_2.A)} \\
Alice_1.B \rightarrow Alice_1.Actor & : & \{Alice_1.Na, Alice_1.Nb\}_{pk(Alice_1.Actor)} \\
Alice_1.Actor \rightarrow Alice_1.B & : & \{Alice_1.Nb\}_{pk(Alice_1.B)} \\
Bob_2.A \rightarrow Bob_2.Actor & : & \{Bob_2.Nb\}_{pk(Bob_2.Actor)}
\end{array}
$$

The instantiation of the obtained attack is:

$$
\begin{array}{rcl}
a \rightarrow i & : & \{c_1, a\}_{pk(i)} \\
i(a) \rightarrow b & : & \{c_1, a\}_{pk(b)} \\
b \rightarrow i(a) & : & \{c_1, c_2\}_{pk(i(a))} \\
i \rightarrow a & : & \{c_1, c_2\}_{pk(a)} \\
a \rightarrow i & : & \{c_2\}_{pk(i)} \\
i(a) \rightarrow b & : & \{c_2\}_{pk(b)}
\end{array}
$$

Verdict NSPK

The instantiation of the obtained attack is:

$$
\begin{array}{rcl}
a \rightarrow i & : & \{c_1, a\}_{pk(i)} \\
i(a) \rightarrow b & : & \{c_1, a\}_{pk(b)} \\
b \rightarrow i(a) & : & \{c_1, c_2\}_{pk(i(a))} \\
i \rightarrow a & : & \{c_1, c_2\}_{pk(a)} \\
a \rightarrow i & : & \{c_2\}_{pk(i)} \\
i(a) \rightarrow b & : & \{c_2\}_{pk(b)}
\end{array}
$$

That is the usual MITM attack on NSPK protocol:

$$
\begin{array}{rcl}
A \rightarrow i & : & \{N_A, A\}_{pk(i)} \\
i(A) \rightarrow B & : & \{N_A, A\}_{pk(B)} \\
B \rightarrow i(A) & : & \{N_A, N_b\}_{pk(A)} \\
i \rightarrow A & : & \{N_A, N_B\}_{pk(A)} \\
A \rightarrow i & : & \{N_B\}_{pk(i)} \\
i(A) \rightarrow B & : & \{N_B\}_{pk(B)}
\end{array}
$$

Outline

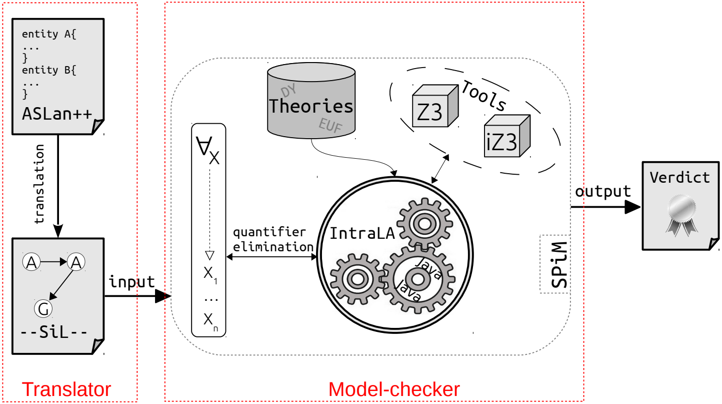
REGIS research group in information security

1. The idea
2. SPiM
   - Method description
   - Example
3. SPiM Java prototype
4. Future work

SPiM

Details

- based on Z3 (sat check) and iZ3 (interpolant generation)
- uses a modified version of IntraLA



Results

A comparison

In order to show the **effectiveness** of our interpolation-based technique, we let the tool run in two different modalities on a few case studies:

1. **IntraLA**: annotation-driven symbolic execution;
2. **Full-explore**: standard symbolic execution (i.e., full exploration of the graph).

Specification (sessions)	IntraLA (# Decide+Learn)	Full-explore (# Decide)	attack
NSL (ab,ab)	125m22s (327+218)	419m15s (587)	no
NSL (ai,ab)	3m15s (81+20)	4m4s (109)	no
NSPK (ab,ab)	54m13s (237+218)	131m53s (587)	no
NSPK (ai,ab)	1m49s (92+20)	1m55s (113)	yes
Helsinki (ab,ab)	224m21s (291+258)	549m38s (681)	no
Yahalom (abs)	22m56s (31)	23m10s (31)	no

Outline

1 The idea

2 SPiM
- Method description
- Example

3 SPiM Java prototype

4 Future work

Future work

- **Full implementation (and more case studies)**
- ASLan++ full coverage
- More complex protocols and goals (LTL)
- **Test case generation and integration in testing phase**

Thank you

Inference rules for a Dolev-Yao intruder

$$\frac{M \in IK}{M \in \mathcal{DY}(IK)}\ G_{\mathrm{axiom}}$$

$$\frac{M_1 \in \mathcal{DY}(IK) \quad M_2 \in \mathcal{DY}(IK)}{[M_1, M_2] \in \mathcal{DY}(IK)}\ G_{\mathrm{pair}} \qquad \frac{[M_1, M_2] \in \mathcal{DY}(IK)}{M_i \in \mathcal{DY}(IK)}\ A_{\mathrm{pair}_i}$$

$$\frac{M_1 \in \mathcal{DY}(IK) \quad M_2 \in \mathcal{DY}(IK)}{\{M_1\}_{M_2} \in \mathcal{DY}(IK)}\ G_{\mathrm{crypt}}$$

$$\frac{\{M_1\}_{M_2} \in \mathcal{DY}(IK) \quad inv(M_2) \in \mathcal{DY}(IK)}{M_1 \in \mathcal{DY}(IK)}\ A_{\mathrm{crypt}}$$

$$\frac{\{M_1\}_{inv(M_2)} \in \mathcal{DY}(IK) \quad M_2 \in \mathcal{DY}(IK)}{M_1 \in \mathcal{DY}(IK)}\ A^{-1}_{\mathrm{crypt}}$$

We convert such a deduction system into a formula (over a finite number of inference steps) and use Z3/iZ3 for performing symbolic execution and calculating annotations.

How we model Dolev-Yao intruder inference

REGIS research group in information security

$$
\begin{aligned}
\varphi_j = \quad & \forall M. (\mathcal{DY}_{IK}^{j+1}(M) \leftrightarrow (\mathcal{DY}_{IK}^{j}(M) \\
& \vee (\exists M'. \mathcal{DY}_{IK}^{j}([M, M']) \vee \mathcal{DY}_{IK}^{j}([M', M])) \\
& \vee (\exists M_1, M_2. M = [M_1, M_2] \wedge \mathcal{DY}_{IK}^{j}(M_1) \wedge \mathcal{DY}_{IK}^{j}(M_2)) \\
& \vee (\exists M_1, M_2. M = \{M_1\}_{M_2} \wedge \mathcal{DY}_{IK}^{j}(M_1) \wedge \mathcal{DY}_{IK}^{j}(M_2))) \\
& \vee (\exists M'. \mathcal{DY}_{IK}^{j}(\{M\}_{M'}) \wedge \mathcal{DY}_{IK}^{j}(inv(M'))) \\
& \vee (\exists M'. \mathcal{DY}_{IK}^{j}(\{M\}_{inv(M')}) \wedge \mathcal{DY}_{IK}^{j}(M'))
\end{aligned}
$$