Proving Cryptographic Protocols with Squirrel
Part 2: Squirrel

David Baelde & Joseph Lallemand

ENS Rennes, IRISA

What is Squirrel?

A proof assistant for
verifying cryptographic protocols,
based on the CCSA approach.

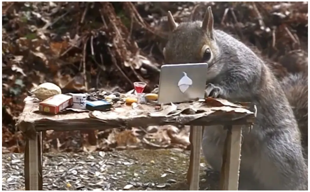
Bana & Comon. *A Computationally Complete Symbolic Attacker for Equivalence Properties.* CCS 2014.

Developped by a group of 7 permanent researchers and 4 PhD students in Rennes, Paris and Nancy.

This talk

An informal introduction to the Squirrel system:

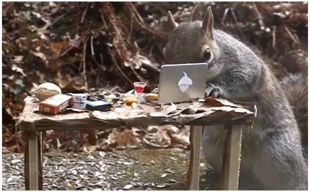
- Preparing the ground for hands-on learning!
- How to formally model protocols and reason about their properties.
- Limited to trace properties: no equivalences.

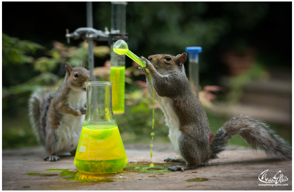


This talk

An informal introduction to the Squirrel system:

- Preparing the ground for hands-on learning!
- How to formally model protocols and reason about their properties.
- Limited to trace properties: no equivalences.





I'm not going to talk about the theory, open problems, related works. . .

Demo

Proving basic logical facts in Squirrel:

`0-logic.sp`

Squirrel uses standard proof assistant UI, and is inspired by Coq.
We prove formulas by organizing them in *sequents*:

$\phi_1, \ldots, \phi_n \vdash \psi$ reads as $(\bigwedge_i \phi_i) \Rightarrow \psi$

The concrete notation is as follows, with identifiers for hypotheses:

```
H_1 : phi_1
...
H_n : phi_n
--------------
psi
```

Demo

Proving basic logical facts in Squirrel:

0-logic.sp

Squirrel uses standard proof assistant UI, and is inspired by Coq.
We prove formulas by organizing them in *sequents*:

$$\phi_1, \ldots, \phi_n \vdash \psi \qquad \text{reads as} \qquad (\bigwedge_i \phi_i) \Rightarrow \psi$$

The concrete notation is as follows, with identifiers for hypotheses:

```
H_1 : phi_1
...
H_n : phi_n
--------------
psi
```

However the Squirrel logic is not as standard as it seems.

Outline

Modelling messages

Crypto is all about probabilistic, polynomial-time (PPTIME) computations.
Reasoning about these directly is intimidating.

Key idea #1

Let's use logical terms to denote PPTIME bitstring computations.

Modelling messages

Crypto is all about probabilistic, polynomial-time (PPTIME) computations.
Reasoning about these directly is intimidating.

Key idea #1

Let's use logical terms to denote PPTIME bitstring computations.

Honest function symbols are interpreted as deterministic computations, used to represent primitives, public constants, etc.
Notation: $\mathsf{f}(m)$, $\mathsf{g}(m, n)$, ok...

Modelling messages

Crypto is all about probabilistic, polynomial-time (PPTIME) computations.
Reasoning about these directly is intimidating.

Key idea #1

Let's use logical terms to denote PPTIME bitstring computations.

Honest function symbols are interpreted as deterministic computations, used to represent primitives, public constants, etc.

Notation: $\mathsf{f}(m)$, $\mathsf{g}(m, n)$, ok...

We assume builtins with standard semantics: equals, ifthenelse, etc.

Example

- $\mathsf{if}\ u = v\ \mathsf{then}\ (\mathsf{if}\ v = u\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2)\ \mathsf{else}\ t_3$ and $\mathsf{if}\ u = v\ \mathsf{then}\ t_1\ \mathsf{else}\ t_3$ always compute the same thing.

Modelling messages

Crypto is all about probabilistic, polynomial-time (PPTIME) computations.
Reasoning about these directly is intimidating.

Key idea #1

Let's use logical terms to denote PPTIME bitstring computations.

Honest function symbols are interpreted as deterministic computations, used to represent primitives, public constants, etc.
Notation: $\mathsf{f}(m)$, $\mathsf{g}(m, n)$, ok...
We assume builtins with standard semantics: equals, ifthenelse, etc.

Example

- $\mathsf{if}\ u = v\ \mathsf{then}\ (\mathsf{if}\ v = u\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2)\ \mathsf{else}\ t_3$ and $\mathsf{if}\ u = v\ \mathsf{then}\ t_1\ \mathsf{else}\ t_3$ always compute the same thing.
- g^x might denote a DH public key associated to private key x.
  To model x, we need probabilistic symbols.

Modelling messages: names

Names

Interpreted as independent uniform random samplings of length $\approx \eta$.
Notation: n, r, k...

Names are used to model private keys, DH exponents, nonces, etc.

Example

- When m and n are distinct name symbols,
  there is a negligible probability that m and n yield the same result.

Modelling messages: names

Names

Interpreted as independent uniform random samplings of length $\approx \eta$.
Notation: n, r, k. . .

Names are used to model private keys, DH exponents, nonces, etc.

Example

- When m and n are distinct name symbols, there is a negligible probability that m and n yield the same result.
- There is a negligible probability that $t =$ n returns true, provided that t represents a computation that cannot use n.
  - $\rightsquigarrow$ This is guaranteed if t contains neither n nor variables. Variables $x, y, z \ldots$ represent *arbitrary* probabilistic computations.

Modelling messages: names

Names

Interpreted as independent uniform random samplings of length $\approx \eta$.
Notation: n, r, k...

Names are used to model private keys, DH exponents, nonces, etc.

Key idea #2

A formula is valid when it is true with overwhelming probability.

Example

- The formula $\mathsf{n} \neq \mathsf{m}$ is valid.
- The formula $t \neq \mathsf{n}$ is valid for any t containing neither n nor variables.

Modelling messages: names

Names

Interpreted as independent uniform random samplings of length $\approx \eta$.
Notation: n, r, k...

Names are used to model private keys, DH exponents, nonces, etc.

Key idea #2

A formula is valid when it is true with overwhelming probability.

Example

- The formula $\mathsf{n} \neq \mathsf{m}$ is valid.
- The formula $t \neq \mathsf{n}$ is valid for any t containing neither n nor variables.
- However, the formula $(\forall x.\ x \neq \mathsf{n})$ is not valid.

Modelling messages: names

Names

Interpreted as independent uniform random samplings of length $\approx \eta$.
Notation: n, r, k...

Names are used to model private keys, DH exponents, nonces, etc.

Key idea #2

A formula is valid when it is true with overwhelming probability.

Example

- The formula $\mathsf{n} \neq \mathsf{m}$ is valid.
- The formula $t \neq \mathsf{n}$ is valid for any t containing neither n nor variables.
- However, the formula $(\forall x.\ \ x \neq \mathsf{n})$ is not valid.

Demo: `1-names.sp` with the `fresh` tactic (also for indexed names)

Modelling messages: adversarial function symbols

Key idea #3

Use unspecified function symbols to model attacker computations.

Adversarial function symbols represent PPTIME computations that cannot access honest randomness (names).
Notation: $\mathbf{att}(m_1, \ldots, m_k)$.

Example (Modelling a trace of signed DH protocol)

🙂	$\rightarrow$	👿			:	out_1	$=$	g^{x}
		👿	$\rightarrow$	😊	:	in_2	$=$	$\mathbf{att}(\mathsf{out}_1)$
		👿	$\leftarrow$	😊	:	out_2	$=$	$\langle\, \mathsf{g}^{\mathsf{y}},\ \mathsf{sign}(\langle \mathsf{g}^{\mathsf{y}}, \mathsf{in}_2 \rangle, \mathsf{sk}$😊$)\, \rangle$
🙂	$\leftarrow$	👿			:	in_3	$=$	$\mathbf{att}'(\mathsf{out}_1, \mathsf{out}_2)$

Cryptographic reasoning: Diffie-Hellman

Key idea #4

Reformulate cryptographic assumptions as axiom schemes by viewing terms as attacker computations.

Assume function symbols for a generator g and exponentiation, interpreted in a cyclic group for which we assume CDH.

Example

- Can we have $\mathsf{g}^{\mathsf{a}} = \mathsf{g}^{\mathsf{a} \times \mathsf{b}}$?

Cryptographic reasoning: Diffie-Hellman

Key idea #4

Reformulate cryptographic assumptions as axiom schemes by viewing terms as attacker computations.

Assume function symbols for a generator g and exponentiation, interpreted in a cyclic group for which we assume CDH.

Example

- Can we have $g^{a} = g^{a \times b}$? No.

Cryptographic reasoning: Diffie-Hellman

Key idea #4

Reformulate cryptographic assumptions as axiom schemes by viewing terms as attacker computations.

Assume function symbols for a generator g and exponentiation, interpreted in a cyclic group for which we assume CDH.

Example

- Can we have $\mathsf{g}^{\mathsf{a}} = \mathsf{g}^{\mathsf{a} \times \mathsf{b}}$? No.
- Can we have $\mathsf{g}^{\mathsf{a} \times \mathsf{b}} = \mathbf{att}(\mathsf{g}^{\mathsf{a}}, \mathsf{g}^{\mathsf{b}})$?

Cryptographic reasoning: Diffie-Hellman

Key idea #4

Reformulate cryptographic assumptions as axiom schemes by viewing terms as attacker computations.

Assume function symbols for a generator g and exponentiation, interpreted in a cyclic group for which we assume CDH.

Example

- Can we have $g^{a} = g^{a \times b}$? No.
- Can we have $g^{a \times b} = \mathbf{att}(g^{a}, g^{b})$? No.

Cryptographic reasoning: Diffie-Hellman

Key idea #4

Reformulate cryptographic assumptions as axiom schemes by viewing terms as attacker computations.

Assume function symbols for a generator g and exponentiation, interpreted in a cyclic group for which we assume CDH.

Example

- Can we have $\mathsf{g}^{\mathsf{a}} = \mathsf{g}^{\mathsf{a} \times \mathsf{b}}$? No.
- Can we have $\mathsf{g}^{\mathsf{a} \times \mathsf{b}} = \mathbf{att}(\mathsf{g}^{\mathsf{a}}, \mathsf{g}^{\mathsf{b}})$? No.
- The formula $\mathsf{g}^{\mathsf{a} \times \mathsf{b}} \neq t$ is valid whenever…

Cryptographic reasoning: Diffie-Hellman

Key idea #4

Reformulate cryptographic assumptions as axiom schemes by viewing terms as attacker computations.

Assume function symbols for a generator g and exponentiation, interpreted in a cyclic group for which we assume CDH.

Example

- Can we have $\mathsf{g}^{\mathsf{a}} = \mathsf{g}^{\mathsf{a} \times \mathsf{b}}$? No.
- Can we have $\mathsf{g}^{\mathsf{a} \times \mathsf{b}} = \mathbf{att}(\mathsf{g}^{\mathsf{a}}, \mathsf{g}^{\mathsf{b}})$? No.
- The formula $\mathsf{g}^{\mathsf{a} \times \mathsf{b}} \neq t$ is valid whenever…
  t contains no variable, only contains a as g^{a}, only contains b as g^{b}.

Cryptographic reasoning: Diffie-Hellman

Key idea #4

Reformulate cryptographic assumptions as axiom schemes by viewing terms as attacker computations.

Assume function symbols for a generator g and exponentiation, interpreted in a cyclic group for which we assume CDH.

Example

- Can we have $g^{a} = g^{a \times b}$? No.
- Can we have $g^{a \times b} = \mathbf{att}(g^{a}, g^{b})$? No.
- The formula $g^{a \times b} \neq t$ is valid whenever...
  t contains no variable, only contains a as g^{a}, only contains b as g^{b}.

Demo: `2-cdh.sp` with the `cdh` tactic (also for indexed secrets)
Demo: `2.5-cdh-signed-dh.sp`

Cryptographic reasoning: signatures

Assume function symbols representing an EUF-CMA signature:

$$\mathsf{sign}(m, \mathsf{k}) : \mathsf{message} \qquad \mathsf{verify}(m, s, \mathsf{pub}(\mathsf{k})) : \mathsf{bool}$$

$$\mathsf{verify}\big(m, \mathsf{sign}(m, \mathsf{k}), \mathsf{pub}(\mathsf{k})\big) = \mathsf{true}$$

Cryptographic reasoning: signatures

Assume function symbols representing an EUF-CMA signature:

$$\mathsf{sign}(m, \mathsf{k}) : \mathsf{message} \qquad \mathsf{verify}(m, s, \mathsf{pub}(\mathsf{k})) : \mathsf{bool}$$

$$\mathsf{verify}\big(m, \mathsf{sign}(m, \mathsf{k}), \mathsf{pub}(\mathsf{k})\big) = \mathsf{true}$$

The following axiom scheme is valid:

$$\mathsf{verify}(m, s, \mathsf{pub}(\mathsf{k})) \Rightarrow \bigvee_{m' \in S} m = m'$$

where $S = \{\ m' \mid \mathsf{sign}(m', \mathsf{k}) \text{ occurs in } m, s\}$
and m, s are closed terms only containing k as $\mathsf{pub}(\mathsf{k})$ and $\mathsf{sign}(\_, \mathsf{k})$.

Cryptographic reasoning: signatures

Assume function symbols representing an EUF-CMA signature:

$$\mathsf{sign}(m, \mathsf{k}) : \mathsf{message} \qquad \mathsf{verify}(m, s, \mathsf{pub}(\mathsf{k})) : \mathsf{bool}$$

$$\mathsf{verify}\big(m, \mathsf{sign}(m, \mathsf{k}), \mathsf{pub}(\mathsf{k})\big) = \mathsf{true}$$

The following axiom scheme is valid:

$$\mathsf{verify}(m, s, \mathsf{pub}(\mathsf{k})) \Rightarrow \bigvee_{m' \in S} m = m'$$

where $S = \{\ m' \mid \mathsf{sign}(m', \mathsf{k}) \text{ occurs in } m, s\}$
and m, s are closed terms only containing k as $\mathsf{pub}(\mathsf{k})$ and $\mathsf{sign}(\_, \mathsf{k})$.

In practice, the tactic `euf H` allows to reason on `H : verify(m,s,pk)` to deduce the above axioms and more.

Demo: `2.5-cdh-signed-dh.sp`

Cryptographic reasoning: hash functions

Assume h models a keyed hash functions.

Cryptographic reasoning: hash functions

Assume h models a keyed hash functions.

If h is EUF-CMA secure, we have

$$u = \mathsf{h}(v, \mathsf{k}) \Rightarrow \bigvee_{s \in S} s = v$$

where $S = \{\ s \mid \mathsf{h}(s, \mathsf{k}) \text{ occurs in } u, v\}$
and u, v are variable-free terms where k only occurs as $\mathsf{h}(\_, \mathsf{k})$.

Cryptographic reasoning: hash functions

Assume h models a keyed hash functions.

If h is EUF-CMA secure, we have

$$u = \mathsf{h}(v, \mathsf{k}) \Rightarrow \bigvee_{s \in S} s = v$$

where $S = \{\ s \mid \mathsf{h}(s, \mathsf{k}) \text{ occurs in } u, v\}$
and u, v are variable-free terms where k only occurs as $\mathsf{h}(\_, \mathsf{k})$.

If h is collision-resistant, we have

$$\mathsf{h}(u, \mathsf{k}) = \mathsf{h}(v, \mathsf{k}) \Rightarrow u = v$$

when u, v are variable-free terms where k only occurs as $\mathsf{h}(\_, \mathsf{k})$.

Cryptographic reasoning: hash functions

Assume h models a keyed hash functions.

If h is EUF-CMA secure, we have

$$u = \mathsf{h}(v, \mathsf{k}) \Rightarrow \bigvee_{s \in S} s = v$$

where $S = \{\ s \mid \mathsf{h}(s, \mathsf{k})$ occurs in $u, v\}$
and u, v are variable-free terms where k only occurs as $\mathsf{h}(\_, \mathsf{k})$.

If h is collision-resistant, we have

$$\mathsf{h}(u, \mathsf{k}) = \mathsf{h}(v, \mathsf{k}) \Rightarrow u = v$$

when u, v are variable-free terms where k only occurs as $\mathsf{h}(\_, \mathsf{k})$.

Existential unforgeability implies collision-resistance,
but the `collision` tactic is more convenient than `euf`.

Further notes

What's in the full local logic?

- Equalities, quantification over indices, boolean connectives, etc.
  Can be seen as PPTIME computation because index is finite.

Further notes

What's in the full local logic?

- With quantifications on message, formulas are not PPTIME anymore, but we can refine our axioms and tactics to carry over.

Further notes

What's in the full local logic?

- With quantifications on message, formulas are not PPTIME anymore, but we can refine our axioms and tactics to carry over.
- Function types, polymorphism, higher-order quantification... for modularity/abstraction in proofs. Recursive definitions (next).

Further notes

What's in the full local logic?

- With quantifications on message, formulas are not PPTIME anymore, but we can refine our axioms and tactics to carry over.
- Function types, polymorphism, higher-order quantification... for modularity/abstraction in proofs. Recursive definitions (next).

Stanislas Riou (PhD student) works on automating the local logic using SMT solvers.

Further notes

What's in the full local logic?

- With quantifications on message, formulas are not PPTIME anymore, but we can refine our axioms and tactics to carry over.
- Function types, polymorphism, higher-order quantification… for modularity/abstraction in proofs. Recursive definitions (next).

Stanislas Riou (PhD student) works on automating the local logic using SMT solvers.

What's beyond the local logic?

The global logic is a classical logic over random vars with predicates for:

- overwhelming truth of a local formula,
- indistinguishability between sequences of messages,

Further notes

What's in the full local logic?

- With quantifications on message, formulas are not PPTIME anymore, but we can refine our axioms and tactics to carry over.
- Function types, polymorphism, higher-order quantification... for modularity/abstraction in proofs. Recursive definitions (next).

Stanislas Riou (PhD student) works on automating the local logic using SMT solvers.

What's beyond the local logic?

The global logic is a classical logic over random vars with predicates for:

- overwhelming truth of a local formula,
- indistinguishability between sequences of messages,
- exact truth of a local formula, (non/bi)-deducibility, etc.

Further notes

What's in the full local logic?

- With quantifications on message, formulas are not PPTIME anymore, but we can refine our axioms and tactics to carry over.
- Function types, polymorphism, higher-order quantification... for modularity/abstraction in proofs. Recursive definitions (next).

Stanislas Riou (PhD student) works on automating the local logic using SMT solvers.

What's beyond the local logic?

The global logic is a classical logic over random vars with predicates for:

- overwhelming truth of a local formula,
- indistinguishability between sequences of messages,
- exact truth of a local formula, (non/bi)-deducibility, etc.

Cryptographic tactics are slowly being subsumed by bi-deduction and cryptographic games thanks to the PhD work of Justine Sauvage.

Outline

Systems of actions

A protocol is modelled by a set of actions.
Each action is identified by an indexed action symbol $\mathsf{A}(\vec{i})$.

The semantics of action $\mathsf{A}(\vec{i})$ is given by:

- a local formula describing the executability condition;
- a message term describing its output.

Both can use a special message term $\mathsf{input}@\mathsf{A}(\vec{i})$.

Example (Signed DH with several sessions)

$\mathsf{A}(i)$ = first action of Alice for session i:

- Executes if true.
- Outputs $\mathsf{g}^{\mathsf{x}(i)}$.

Systems of actions

A protocol is modelled by a set of actions.
Each action is identified by an indexed action symbol $\mathsf{A}(\vec{i})$.

The semantics of action $\mathsf{A}(\vec{i})$ is given by:

- a local formula describing the executability condition;
- a message term describing its output.

Both can use a special message term $\mathsf{input@A}(\vec{i})$.

Example (Signed DH with several sessions)

$\mathsf{B}(j)$ = first action of Bob for session j:

- Executes if true.
- Outputs $\langle\ \mathsf{g}^{\mathsf{y}(j)},\ \mathsf{sign}(\langle \mathsf{g}^{\mathsf{y}(j)}, \mathsf{input@B}(j)\rangle, \mathsf{sk}😊)\ \rangle$.

Systems of actions

A protocol is modelled by a set of actions.
Each action is identified by an indexed action symbol $\mathsf{A}(\vec{i})$.

The semantics of action $\mathsf{A}(\vec{i})$ is given by:

- a local formula describing the executability condition;
- a message term describing its output.

Both can use a special message term $\mathsf{input}@\mathsf{A}(\vec{i})$.

Example (Signed DH with several sessions)

$\mathsf{A}_1(i) =$ second action of Alice for session i, upon success:

- Executes if
  $\mathsf{verify}(\ \langle \mathsf{fst}(\mathsf{input}@\mathsf{A}_1(i)), \mathsf{g}^{\mathsf{x}^{(i)}} \rangle,\ \mathsf{snd}(\mathsf{input}@\mathsf{A}_1(i)),\ \mathsf{pub}(\mathsf{sk}😊)\ )$.
- Outputs $\mathsf{sign}(\langle \mathsf{g}^{\mathsf{x}^{(i)}}, \mathsf{fst}(\mathsf{input}@\mathsf{A}_1(i)) \rangle, \mathsf{sk}🙂)$.

Systems of actions

A protocol is modelled by a set of actions.
Each action is identified by an indexed action symbol $\mathsf{A}(\vec{i})$.

The semantics of action $\mathsf{A}(\vec{i})$ is given by:

- a local formula describing the executability condition;
- a message term describing its output.

Both can use a special message term $\mathsf{input}@\mathsf{A}(\vec{i})$.

Example (Signed DH with several sessions)

$\mathsf{A}_2(i)$ = second action of Alice for session i, upon failure:

- Executes if
  $\neg\mathsf{verify}(\ \langle \mathsf{fst}(\mathsf{input}@\mathsf{A}_2(i)), \mathsf{g}^{\mathsf{x}(i)}\rangle,\ \mathsf{snd}(\mathsf{input}@\mathsf{A}_2(i)),\ \mathsf{pub}(\mathsf{sk}\text{😊})\ )$.
- Outputs 👎.

Systems of actions

A protocol is modelled by a set of actions.
Each action is identified by an indexed action symbol $\mathsf{A}(\vec{i})$.

The semantics of action $\mathsf{A}(\vec{i})$ is given by:

- a local formula describing the executability condition;
- a message term describing its output.

Both can use a special message term $\mathsf{input@A}(\vec{i})$.

Example (Signed DH with several sessions)

$\mathsf{A}_2(i)$ = second action of Alice for session i, upon failure:

- Executes if
  $\neg\mathsf{verify}(\ \langle \mathsf{fst}(\mathsf{input@A}_2(i)), \mathsf{g}^{\mathsf{x}(i)}\rangle,\ \mathsf{snd}(\mathsf{input@A}_2(i)),\ \mathsf{pub}(\mathsf{sk}😊)\ )$.
- Outputs 👎.

Demo: `3.5-signed-dh-many.sp` (actions compiled from π-calculus process)

Traces of actions

A trace is a non-repeating sequence of actions subject to protocol-specific conditions:

- $A(1).B(7).A(1)$ is not a trace.
- $A(1).B(7).A_1(1)$ is a trace.

Traces of actions

A trace is a non-repeating sequence of actions subject to protocol-specific conditions:

- $A(1).B(7).A(1)$ is not a trace.
- $A(1).B(7).A_1(1)$ is a trace.
- $A(1).B(7).A_1(3)$ is not a trace.

Traces of actions

A trace is a non-repeating sequence of actions subject to protocol-specific conditions:

- $A(1).B(7).A(1)$ is not a trace.
- $A(1).B(7).A_1(1)$ is a trace.
- $A(1).B(7).A_1(3)$ is not a trace.
- $A(1).B(7).A_2(1)$ is a trace.

Traces of actions

A trace is a non-repeating sequence of actions subject to protocol-specific conditions:

- $A(1).B(7).A(1)$ is not a trace.
- $A(1).B(7).A_1(1)$ is a trace.
- $A(1).B(7).A_1(3)$ is not a trace.
- $A(1).B(7).A_2(1)$ is a trace.
- $A(1).B(7).A_1(1).A_2(1)$ is not a trace.

Traces of actions

A trace is a non-repeating sequence of actions subject to protocol-specific conditions:

- $A(1).B(7).A(1)$ is not a trace.
- $A(1).B(7).A_1(1)$ is a trace.
- $A(1).B(7).A_1(3)$ is not a trace.
- $A(1).B(7).A_2(1)$ is a trace.
- $A(1).B(7).A_1(1).A_2(1)$ is not a trace.
- $B(7).A(1).A_1(1)$ is a trace.

A trace just indicates a tentative schedule for actions. Depending on the interpretation of primitives and attackers, it will execute with a certain probability.

Modelling traces in the logic

We use terms of sort timestamp:

- happens(τ) means that τ is part of the trace
- init is the first timestamp that happens
- $<$ is a total order on timestamps that happen

Each trace yields a trace model, i.e.,
an interpretation for the action symbols, happens and $<$.

Modelling traces in the logic

We use terms of sort timestamp:

- happens(τ) means that τ is part of the trace
- init is the first timestamp that happens
- $<$ is a total order on timestamps that happen

Each trace yields a trace model, i.e.,
an interpretation for the action symbols, happens and $<$.

Injectivity (part of `auto` tactic)

For distinct actions $\mathsf{A}, \mathsf{B} \in \mathcal{A}$:

- $\forall \vec{i}.\ \ \forall \vec{j}.\ \ \mathsf{happens}(\mathsf{A}(\vec{i})) \wedge \mathsf{happens}(\mathsf{B}(\vec{j})) \ \Rightarrow\ \mathsf{A}(\vec{i}) \neq \mathsf{B}(\vec{j})$
- $\forall \vec{i}.\ \ \forall \vec{j}.\ \ \mathsf{happens}(\mathsf{A}(\vec{i})) \wedge \mathsf{happens}(\mathsf{A}(\vec{j})) \wedge \vec{i} \neq \vec{j} \ \Rightarrow\ \mathsf{A}(\vec{i}) \neq \mathsf{A}(\vec{j})$

Modelling traces in the logic

We use terms of sort timestamp:

- happens(τ) means that τ is part of the trace
- init is the first timestamp that happens
- $<$ is a total order on timestamps that happen

Each trace yields a trace model, i.e.,
an interpretation for the action symbols, happens and $<$.

Order (part of `auto` tactic)

- $\mathsf{happens}(\tau) \wedge \mathsf{happens}(\tau') \Leftrightarrow \tau \leq \tau' \vee \tau' \leq \tau$
- $\mathsf{happens}(\mathsf{pred}(\tau)) \Rightarrow \mathsf{pred}(\tau) < \tau$

Modelling traces in the logic

We use terms of sort timestamp:

- happens(τ) means that τ is part of the trace
- init is the first timestamp that happens
- $<$ is a total order on timestamps that happen

Each trace yields a trace model, i.e.,
an interpretation for the action symbols, happens and $<$.

Case analysis and induction (`case` and `induction`)

- $\forall \tau.\ \ \mathsf{happens}(\tau) \Rightarrow \tau = \mathsf{init} \vee \bigvee_{\mathsf{A} \in \mathcal{A}} \exists \vec{i}.\ \ \tau = A(\vec{i})$
- $(\forall\ \tau.\ (\forall \tau'.\ \ \tau' < \tau \Rightarrow\ \ \phi[\tau']) \Rightarrow \phi[\tau]) \Rightarrow \forall \tau.\ \ \phi[\tau]$

Modelling traces in the logic

We use terms of sort timestamp:

- happens(τ) means that τ is part of the trace
- init is the first timestamp that happens
- $<$ is a total order on timestamps that happen

Each trace yields a trace model, i.e.,
an interpretation for the action symbols, happens and $<$.

Signed DH specific axioms (used by `smt` but not `auto`)

- $\forall i.\ \mathsf{happens}(\mathsf{A}_1(i)) \Rightarrow \mathsf{A}(i) < \mathsf{A}_1(i)$ (dependency)
- $\forall i.\ \neg\big(\mathsf{happens}(\mathsf{A}_1(i)) \wedge \mathsf{happens}(\mathsf{A}_2(i))\big)$ (conflict)

Macros

Given a trace we define[1] recursively several macros encoding the attacker's interaction with the protocol along that trace:

$$\begin{array}{lcll} \mathsf{output}@\tau & = & \langle\text{output of action } \tau\rangle & \text{if } \mathsf{init} < \tau \\ \mathsf{cond}@\tau & = & \langle\text{condition of action } \tau\rangle & \text{if } \mathsf{init} < \tau \end{array}$$

[1] Missing cases are not important.

Macros

Given a trace we define[1] recursively several macros encoding the attacker's interaction with the protocol along that trace:

$$
\begin{array}{lcll}
\mathsf{output}@\tau & = & \langle \text{output of action } \tau \rangle & \text{if } \mathsf{init} < \tau \\
\mathsf{cond}@\tau & = & \langle \text{condition of action } \tau \rangle & \text{if } \mathsf{init} < \tau \\
\mathsf{exec}@\mathsf{init} & = & \mathsf{true} & \\
\mathsf{exec}@\tau & = & \mathsf{exec}@\mathsf{pred}(\tau) \wedge \mathsf{cond}@\tau & \text{if } \mathsf{init} < \tau
\end{array}
$$

[1] Missing cases are not important.

Macros

Given a trace we define[1] recursively several macros encoding the attacker's interaction with the protocol along that trace:

$$
\begin{array}{lcll}
\mathsf{output}@\tau & = & \langle\text{output of action } \tau\rangle & \text{if } \mathsf{init} < \tau \\
\mathsf{cond}@\tau & = & \langle\text{condition of action } \tau\rangle & \text{if } \mathsf{init} < \tau \\
\mathsf{exec}@\mathsf{init} & = & \mathsf{true} & \\
\mathsf{exec}@\tau & = & \mathsf{exec}@\mathsf{pred}(\tau) \wedge \mathsf{cond}@\tau & \text{if } \mathsf{init} < \tau \\
\mathsf{frame}@\mathsf{init} & = & \mathsf{empty} & \\
\mathsf{exec}@\tau & = & \langle\ \mathsf{frame}@\mathsf{pred}(\tau), & \\
 & & \quad \langle\ \mathsf{exec}@\tau, & \\
 & & \quad\quad \mathsf{if}\ \mathsf{exec}@\tau\ \mathsf{then}\ \mathsf{output}@\tau\ \mathsf{else}\ \mathsf{empty}\ \rangle\rangle & \text{if } \mathsf{init} < \tau
\end{array}
$$

[1] Missing cases are not important.

Macros

Given a trace we define[1] recursively several macros encoding the attacker's interaction with the protocol along that trace:

$$
\begin{array}{lcll}
\mathsf{output}@\tau & = & \langle \text{output of action } \tau \rangle & \text{if init} < \tau \\
\mathsf{cond}@\tau & = & \langle \text{condition of action } \tau \rangle & \text{if init} < \tau \\
\mathsf{exec}@\mathsf{init} & = & \mathsf{true} & \\
\mathsf{exec}@\tau & = & \mathsf{exec}@\mathsf{pred}(\tau) \wedge \mathsf{cond}@\tau & \text{if init} < \tau \\
\mathsf{frame}@\mathsf{init} & = & \mathsf{empty} & \\
\mathsf{exec}@\tau & = & \langle\ \mathsf{frame}@\mathsf{pred}(\tau), & \\
 & & \quad \langle\ \mathsf{exec}@\tau, & \\
 & & \qquad \text{if } \mathsf{exec}@\tau \text{ then } \mathsf{output}@\tau \text{ else empty } \rangle\rangle & \text{if init} < \tau \\
\mathsf{input}@\tau & = & \mathbf{att}(\mathsf{frame}@\tau) & \text{if init} < \tau
\end{array}
$$

[1]Missing cases are not important.

Security properties for all traces

Additional macros reflect let-definitions used in processes.
For example $\mathsf{Y}'@\mathsf{A}_1(i)$ is the value of Y' in that action.

Example (Agreement for 🙂)

$$\begin{aligned}\forall i.\ \ \mathsf{cond}@\mathsf{A}_1(i) \Rightarrow \exists j.\ & \mathsf{B}(j) < \mathsf{A}_1(i) \wedge \\ & \mathsf{X}@\mathsf{A}(i) = \mathsf{X}'@\mathsf{B}(j) \wedge \\ & \mathsf{Y}'@\mathsf{A}_1(i) = \mathsf{Y}@\mathsf{B}(j)\end{aligned}$$

Demo: `3.5-signed-dh-many.sp`

Reasoning with recursive definitions

Constraining occurrences becomes more complex with macros.

Example (Freshness without macros nor indices)

$t \neq \mathsf{n}$ is valid for any variable-free term t that does not contain n

Reasoning with recursive definitions

Constraining occurrences becomes more complex with macros.

Example (Freshness without macros nor indices)

$t \neq \mathsf{n}$ is valid for any variable-free term t that does not contain n

Example (Freshness without macros)

$t = \mathsf{n}(\vec{i}) \Rightarrow \bigvee_{\mathsf{n}(\vec{j}) \in t} \vec{i} = \vec{j}$ valid for any term t without message variables

Reasoning with recursive definitions

Constraining occurrences becomes more complex with macros.

Example (Freshness without macros nor indices)

$t \neq \mathsf{n}$ is valid for any variable-free term t that does not contain n

Example (Freshness without macros)

$t = \mathsf{n}(\vec{i}) \Rightarrow \bigvee_{\mathsf{n}(\vec{j}) \in t} \vec{i} = \vec{j}$ valid for any term t without message variables

Example (Freshness with macros)

$$t = \mathsf{n}(\vec{i}) \quad \Rightarrow \quad \bigvee_{\mathsf{n}(\vec{j}) \in t} \vec{i} = \vec{j} \quad \vee \quad \bigvee_{\mathsf{n}(\vec{j}) \in \mathsf{A}(\vec{k})} \exists \vec{k}.\ \ \mathsf{A}(\vec{k}) \leq T \wedge \vec{i} = \vec{j}$$

valid for any term t without message variables,
where $\mathsf{n}(j) \in \mathsf{A}(\vec{k})$ denotes occurrences in output or condition of A

Reasoning with recursive definitions

Constraining occurrences becomes more complex with macros.

Example (Freshness without macros nor indices)

$t \neq \mathsf{n}$ is valid for any variable-free term t that does not contain n

Example (Freshness without macros)

$t = \mathsf{n}(\vec{i}) \Rightarrow \bigvee_{\mathsf{n}(\vec{j}) \in t} \vec{i} = \vec{j}$ valid for any term t without message variables

Example (Freshness with macros)

$$t = \mathsf{n}(\vec{i}) \quad \Rightarrow \quad \bigvee_{\mathsf{n}(\vec{j}) \in t} \vec{i} = \vec{j} \quad \vee \quad \bigvee_{\mathsf{n}(\vec{j}) \in \mathsf{A}(\vec{k})} \exists \vec{k}.\ \mathsf{A}(\vec{k}) \leq T \wedge \vec{i} = \vec{j}$$

valid for any term t without message variables,
where $\mathsf{n}(j) \in \mathsf{A}(\vec{k})$ denotes occurrences in output or condition of A

Further refinements are possible and even necessary in practice.

Further notes

Protocols with mutable state

Protocols with mutable memory cells are supported (using cell@τ macros). The translation from processes to systems of actions, and its soundness, is recent work notably involving Clément Herouard.

Further notes

Protocols with mutable state

Protocols with mutable memory cells are supported (using cell@τ macros). The translation from processes to systems of actions, and its soundness, is recent work notably involving Clément Herouard.

Polynomial security

A subtle discrepancy between security notions:

- We prove that, for each trace T, there is no attacker along T.
- We would like to prove that there is no attacker, choosing the trace depending on η and previous messages.

This is the topic of Théo Vigneron's ongoing PhD thesis.

What's next?

Hands on experience in practical sessions!

Learn some more on our website, with more tutorials and papers:

https://squirrel-prover.github.io/