Automated and machine-verified security proofs of stateful protocols

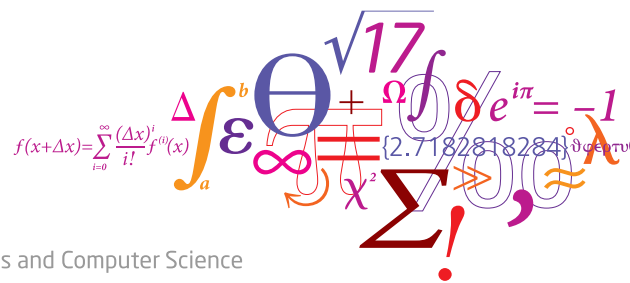
Andreas Hess[1] Sebastian Mödersheim[1]

Achim Brucker[2,3] Anders Schlichtkrull[1]

[1]Technical University of Denmark

[2]The University of Sheffield

[3]University of Exeter



DTU Compute
Department of Applied Mathematics and Computer Science

Overview

1. Stateful protocol verification
2. What we are doing
3. Demo
4. Conclusion

Example: A Keyserver

Stateful protocols:

- Global mutable state spanning multiple sessions

Scenario:

- A server maintains a database of public keys for users
  - Set valid(A) of valid keys of user A
  - Set revoked(A) of revoked keys of user A
- Each user A has a keyring ring(A).

Example: A Keyserver

```
outOfBand(A:honest)
 new PK
 insert PK ring(A)
 insert PK valid(A)
 send PK.
```

Joint transaction between an agent A and the keyserver.

Example: A Keyserver

```
updateKey(A:honest,PK:value)
 PK in ring(A)
 new NPK
 delete PK ring(A)
 insert NPK ring(A)
 send sign(inv(PK),NPK).
```

```
updateKeyServer(A:honest,PK:value,NPK:value)
 receive sign(inv(PK),NPK)
 PK in valid(A)
 NPK notin valid(A)
 NPK notin revoked(A)
 delete PK valid(A)
 insert PK revoked(A)
 insert NPK valid(A).
```

Example: A Keyserver

```
oopsEvent(A:honest,PK:value)
 PK in revoked(A)
 send inv(PK).
```

```
authAttack(A:honest,PK:value)
 receive inv(PK)
 PK in valid(A)
 attack.
```

There is an attack if there exists a run of the protocol in which the `authAttack` transaction fires

Over-Approximation

Popular approach in protocol verification: Ask the question:
"What messages can the intruder ever learn in any reachable state?"

To keep things decidable we over-approximate and restrict the intruder to a typed model

- Not all abstract states are feasible in the real world,
  - but we are on the safe side (it is a sound over-approximation).

$$\begin{aligned}
&\mathsf{pk}_1, \mathsf{pk}_2, \ldots \\
&\mathsf{sign}(\mathsf{inv}(\mathsf{pk}'_1), \mathsf{npk}_1), \mathsf{sign}(\mathsf{inv}(\mathsf{pk}'_2), \mathsf{npk}_2), \ldots \\
&\mathsf{inv}(\mathsf{pk}''_1), \mathsf{inv}(\mathsf{pk}''_2), \ldots \\
&\mathsf{npk}_1, \mathsf{npk}_2, \ldots
\end{aligned}$$

Over-Approximation

Popular approach in protocol verification: Ask the question:
"What messages can the intruder ever learn in any reachable state?"

To keep things decidable we over-approximate and restrict the intruder to a typed model

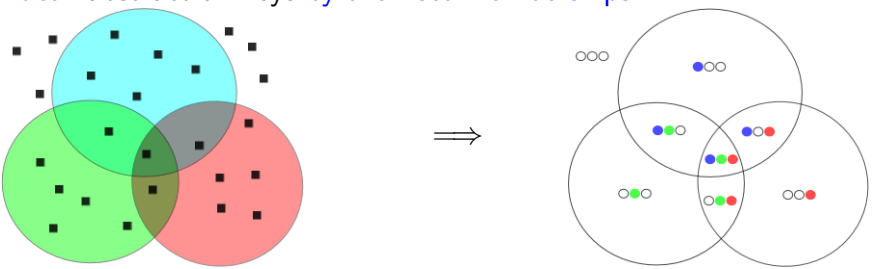
- Not all abstract states are feasible in the real world,
  - but we are on the safe side (it is a sound over-approximation).

$$PK,$$
$$\mathsf{sign}(\mathsf{inv}(PK'), NPK),$$
$$\mathsf{inv}(PK'')$$

Set-Based Abstraction

How do we handle the databases?

Idea: abstract all keys by their set memberships.



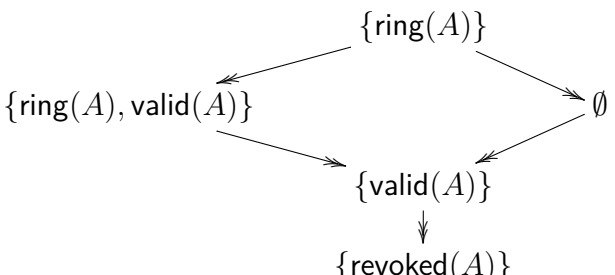
- Implemented in AIF/AIF-ω and Set-π (similar ideas in StatVerif and GSVerif)

Example: The Fixed-Point for the Keyserver

PK,
$\mathsf{sign}(\mathsf{inv}(PK'), NPK)$,
$\mathsf{inv}(PK'')$

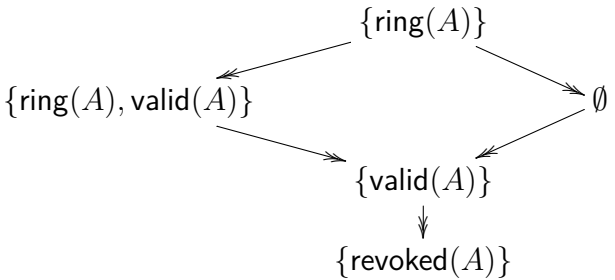
Example: The Fixed-Point for the Keyserver

$\{\mathsf{ring}(A)\}$,
$\mathsf{sign}(\mathsf{inv}(\emptyset), \{\mathsf{ring}(A)\})$,
$\mathsf{inv}(\{\mathsf{revoked}(A)\})$



Example: The Fixed-Point for the Keyserver

$\{\mathsf{ring}(A)\}$,
$\mathsf{sign}(\mathsf{inv}(\emptyset), \{\mathsf{ring}(A)\})$,
$\mathsf{inv}(\{\mathsf{revoked}(A)\})$



- The intruder also knows, e.g.,
  $\mathsf{sign}(\mathsf{inv}(\{\mathsf{valid}(A)\}), \{\mathsf{revoked}(A)\})$
- Since the attack signal does not occur in the fixed-point the keyserver protocol is secure

Should we trust the output of verification tools?

Automatic (OFMC, AIF-ω, GSVerif, ...)	*Interactive* (Isabelle, Coq, Twelf, ...)
May contain bugs $\Rightarrow$ flawed security claims!	Extremely high correctness guarantee
Automated	Requires a lot of expertise
Fast	Time consuming and can be tedious

- Goal: Use automatic methods to obtain a "proof" for proof assistants to check, combining the advantages of both
- Every proof accepted by Isabelle/HOL is machine-verified
  - Every proof argument is verified down to the axioms
  - We only have to trust the small core of Isabelle
  - Subtle assumptions cannot be overlooked

What we are doing

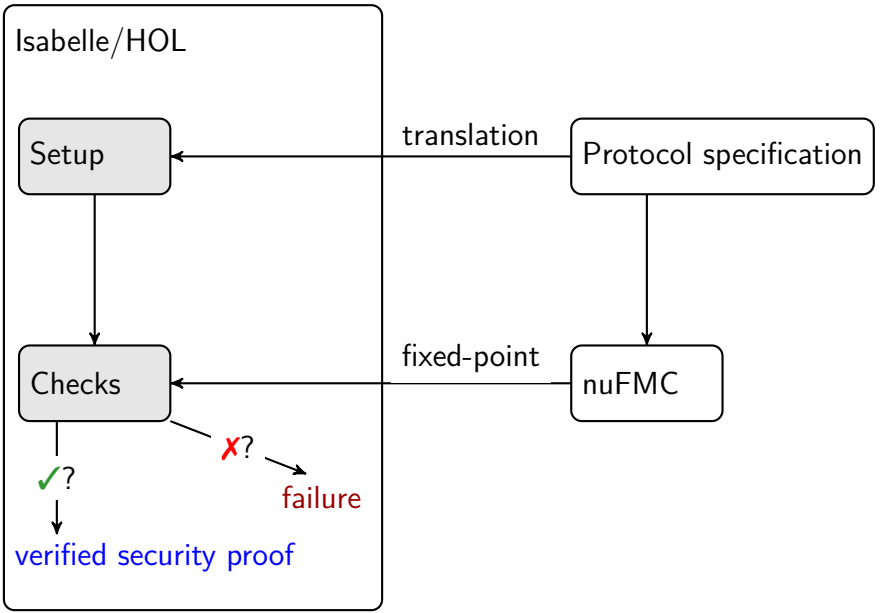
1. Proved a theorem for protocol security in Isabelle. Roughly,

Theorem

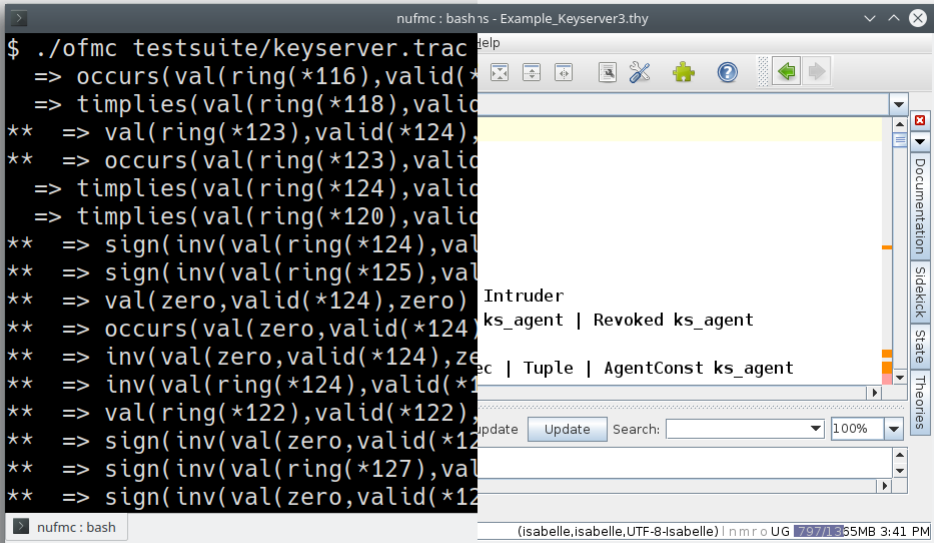
If FP is a fixed-point that covers the protocol $\mathcal{P}$, and the attack *signal does not occur in FP, then $\mathcal{P}$ is secure*

2. The conditions to check falls within the executable fragment of Isabelle
   - They are automatically verifiable
   - The checks terminate (assuming reasonable conditions on the protocol specifications)
3. Extended the OFMC tool to support stateful protocols: nuFMC
4. Connected nuFMC to the Isabelle formalization

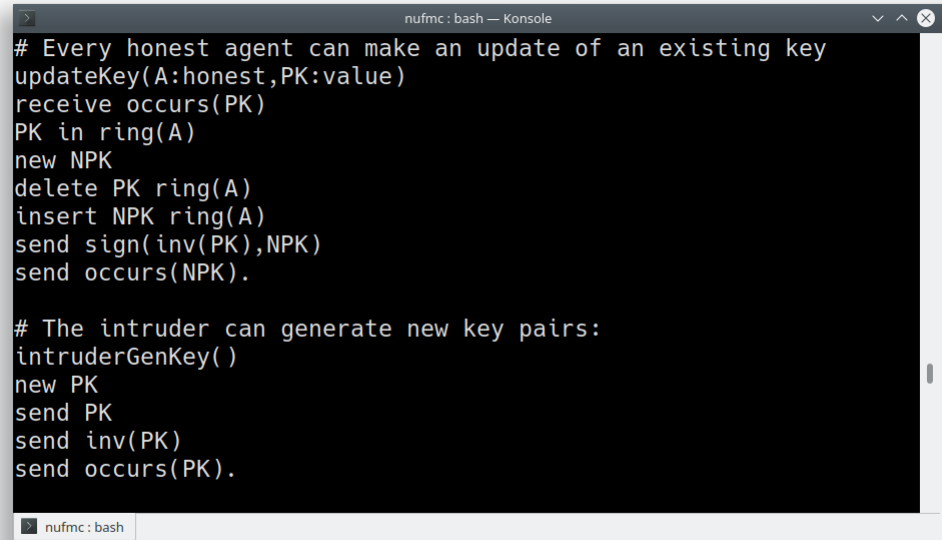
Tool Overview



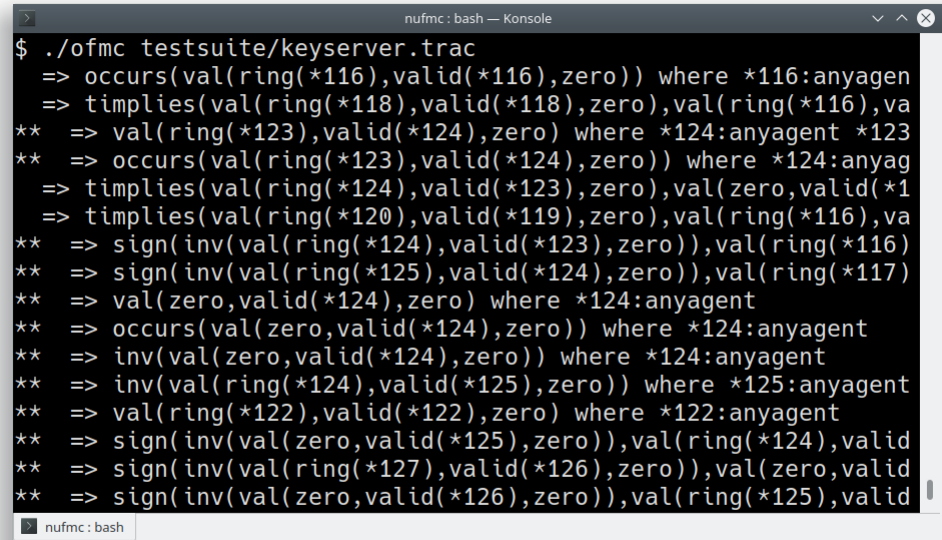
Demo


nufmc : bash
ns - Example_Keyserver3.thy
$ ./ofmc testsuite/keyserver.trac
=> occurs(val(ring(*116),valid(*
=> timplies(val(ring(*118),valid
** => val(ring(*123),valid(*124),
** => occurs(val(ring(*123),valid
=> timplies(val(ring(*124),valid
=> timplies(val(ring(*120),valid
** => sign(inv(val(ring(*124),val
** => sign(inv(val(ring(*125),val
** => val(zero,valid(*124),zero)
** => occurs(val(zero,valid(*124
** => inv(val(zero,valid(*124),ze
** => inv(val(ring(*124),valid(*1
** => val(ring(*122),valid(*122),
** => sign(inv(val(zero,valid(*12
** => sign(inv(val(ring(*127),val
** => sign(inv(val(zero,valid(*12
elp
Intruder
ks_agent | Revoked ks_agent
ac | Tuple | AgentConst ks_agent
pdate
Update
Search:
100%
Documentation
Sidekick
State
Theories
nufmc : bash
(isabelle,isabelle,UTF-8-Isabelle) I n m r o UG 797/1
35MB 3:41 PM

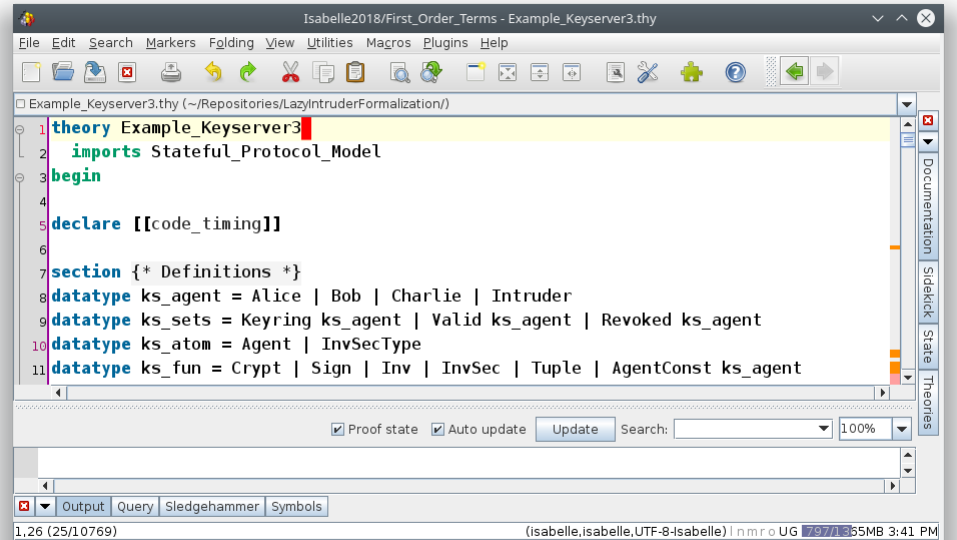
Demo


nufmc : bash — Konsole
# Every honest agent can make an update of an existing key
updateKey(A:honest,PK:value)
receive occurs(PK)
PK in ring(A)
new NPK
delete PK ring(A)
insert NPK ring(A)
send sign(inv(PK),NPK)
send occurs(NPK).
# The intruder can generate new key pairs:
intruderGenKey( )
new PK
send PK
send inv(PK)
send occurs(PK).
nufmc : bash

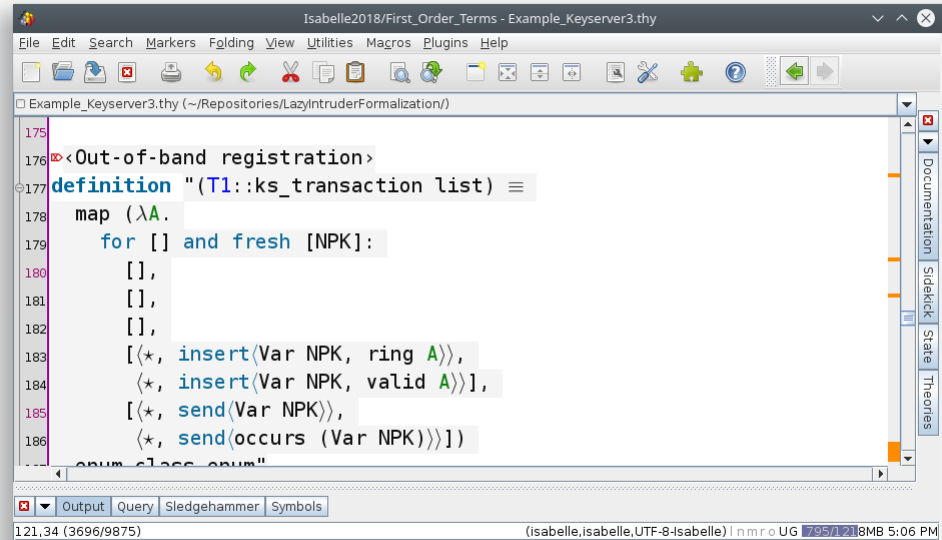
Demo


nufmc : bash — Konsole
$ ./ofmc testsuite/keyserver.trac
=> occurs(val(ring(*116),valid(*116),zero)) where *116:anyagen
=> timplies(val(ring(*118),valid(*118),zero),val(ring(*116),va
** => val(ring(*123),valid(*124),zero) where *124:anyagent *123
** => occurs(val(ring(*123),valid(*124),zero)) where *124:anyag
=> timplies(val(ring(*124),valid(*123),zero),val(zero,valid(*1
=> timplies(val(ring(*120),valid(*119),zero),val(ring(*116),va
** => sign(inv(val(ring(*124),valid(*123),zero)),val(ring(*116)
** => sign(inv(val(ring(*125),valid(*124),zero)),val(ring(*117)
** => val(zero,valid(*124),zero) where *124:anyagent
** => occurs(val(zero,valid(*124),zero)) where *124:anyagent
** => inv(val(zero,valid(*124),zero)) where *124:anyagent
** => inv(val(ring(*124),valid(*125),zero)) where *125:anyagent
** => val(ring(*122),valid(*122),zero) where *122:anyagent
** => sign(inv(val(zero,valid(*125),zero)),val(ring(*124),valid
** => sign(inv(val(ring(*127),valid(*126),zero)),val(zero,valid
** => sign(inv(val(zero,valid(*126),zero)),val(ring(*125),valid
nufmc : bash

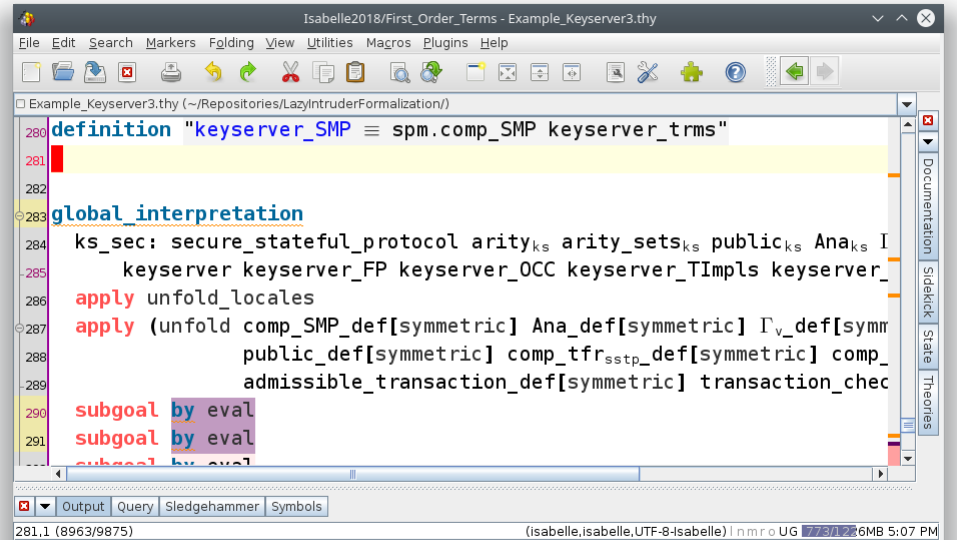
Demo


Isabelle2018/First_Order_Terms - Example_Keyserver3.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Example_Keyserver3.thy (~/Repositories/LazyIntruderFormalization/)
theory Example_Keyserver3
imports Stateful_Protocol_Model
begin
declare [[code_timing]]
section {* Definitions *}
datatype ks_agent = Alice | Bob | Charlie | Intruder
datatype ks_sets = Keyring ks_agent | Valid ks_agent | Revoked ks_agent
datatype ks_atom = Agent | InvSecType
datatype ks_fun = Crypt | Sign | Inv | InvSec | Tuple | AgentConst ks_agent
Documentation Sidekick State Theories
Proof state
Auto update
Update
Search:
100%
Output Query Sledgehammer Symbols
1,26 (25/10769)
(isabelle,isabelle,UTF-8-Isabelle)
3:41 PM

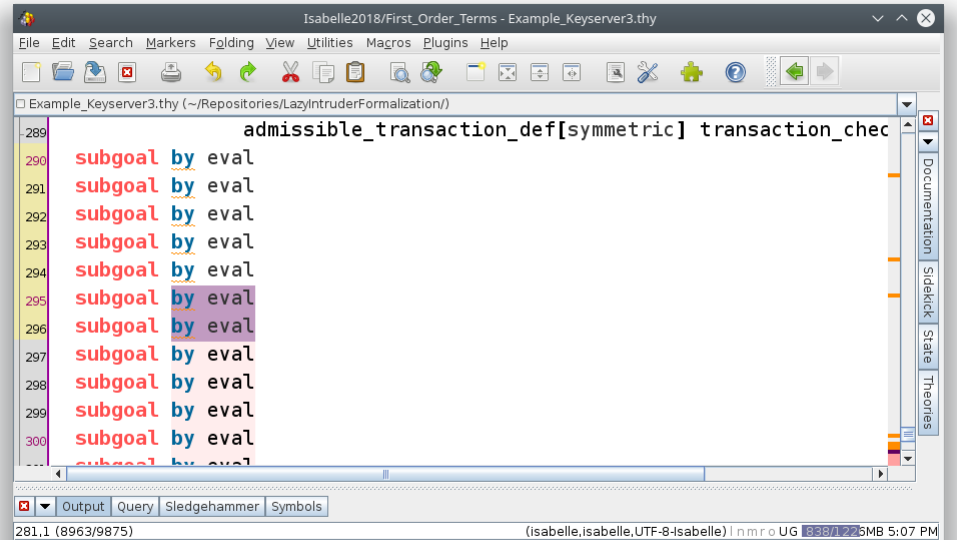
Demo


Isabelle2018/First_Order_Terms - Example_Keyserver3.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Example_Keyserver3.thy (~/Repositories/LazyIntruderFormalization/)
175
176 ‹Out-of-band registration›
177 definition "(T1::ks_transaction list) ≡
178 map (λA.
179 for [] and fresh [NPK]:
180 [],
181 [],
182 [],
183 [⟨⋆, insert⟨Var NPK, ring A⟩⟩,
184 ⟨⋆, insert⟨Var NPK, valid A⟩⟩],
185 [⟨⋆, send⟨Var NPK⟩⟩,
186 ⟨⋆, send⟨occurs (Var NPK)⟩⟩])
Documentation
Sidekick
State
Theories
Output
Query
Sledgehammer
Symbols
121,34 (3696/9875)
(isabelle,isabelle,UTF-8-Isabelle)
8MB 5:06 PM

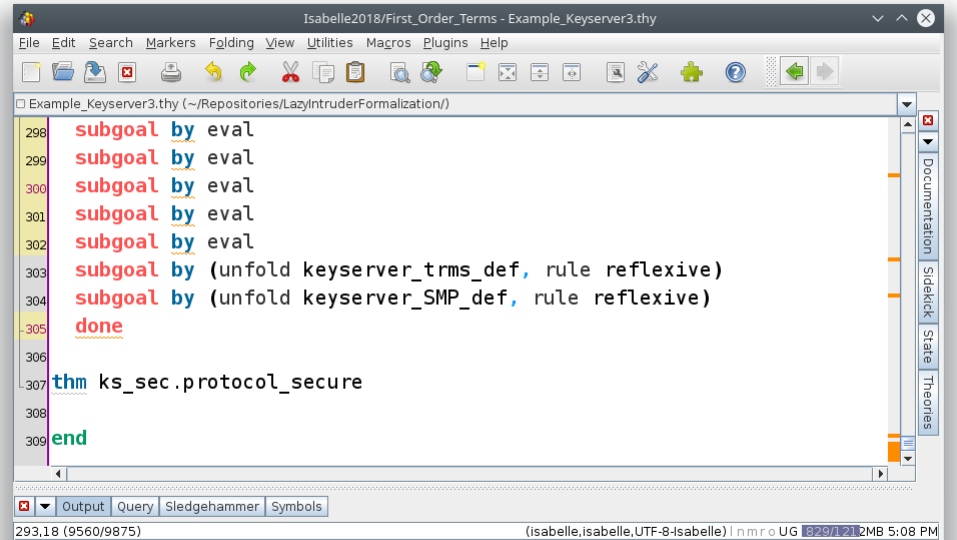
Demo


Isabelle2018/First_Order_Terms - Example_Keyserver3.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Example_Keyserver3.thy (~/Repositories/LazyIntruderFormalization/)
280 definition "keyserver_SMP ≡ spm.comp_SMP keyserver_trms"
281
282
283 global_interpretation
284 ks_sec: secure_stateful_protocol arity$_{ks}$ arity_sets$_{ks}$ public$_{ks}$ Ana$_{ks}$ I
285 keyserver keyserver_FP keyserver_OCC keyserver_TImpls keyserver_
286 apply unfold_locales
287 apply (unfold comp_SMP_def[symmetric] Ana_def[symmetric] Γ$_v$_def[symm
288 public_def[symmetric] comp_tfr$_{sstp}$_def[symmetric] comp_
289 admissible_transaction_def[symmetric] transaction_chec
290 subgoal by eval
291 subgoal by eval
Documentation
Sidekick
State
Theories
Output
Query
Sledgehammer
Symbols
281,1 (8963/9875)
(isabelle,isabelle,UTF-8-Isabelle) I n m r o UG 773/1226MB 5:07 PM

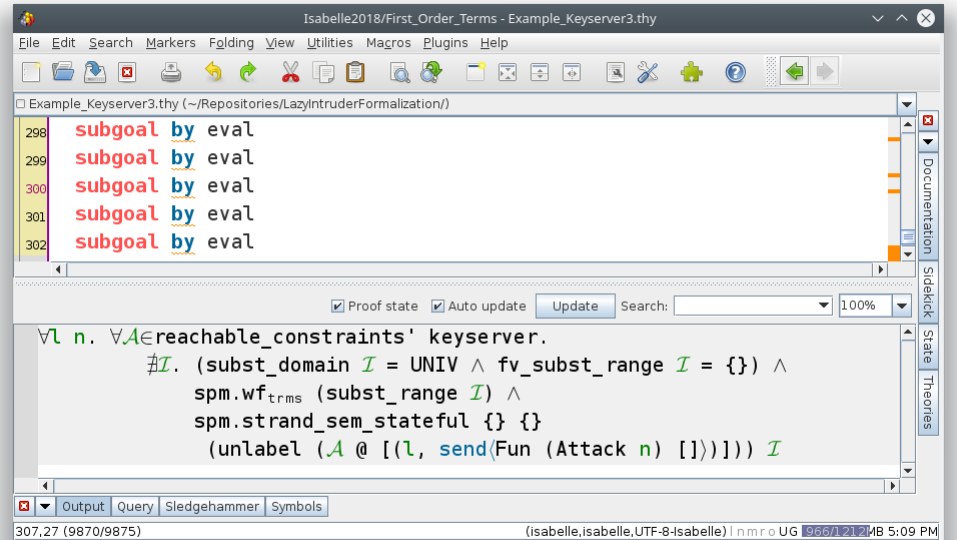
Demo


Isabelle2018/First_Order_Terms - Example_Keyserver3.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Example_Keyserver3.thy (~/Repositories/LazyIntruderFormalization/)
admissible_transaction_def[symmetric] transaction_chec
subgoal by eval
subgoal by eval
subgoal by eval
subgoal by eval
subgoal by eval
subgoal by eval
subgoal by eval
subgoal by eval
subgoal by eval
subgoal by eval
subgoal by eval
Documentation
Sidekick
State
Theories
Output
Query
Sledgehammer
Symbols
281,1 (8963/9875)
(isabelle,isabelle,UTF-8-Isabelle)
UG
5:07 PM

Demo


Isabelle2018/First_Order_Terms - Example_Keyserver3.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Example_Keyserver3.thy (~/Repositories/LazyIntruderFormalization/)
298 subgoal by eval
299 subgoal by eval
300 subgoal by eval
301 subgoal by eval
302 subgoal by eval
303 subgoal by (unfold keyserver_trms_def, rule reflexive)
304 subgoal by (unfold keyserver_SMP_def, rule reflexive)
305 done
306
307 thm ks_sec.protocol_secure
308
309 end
Documentation
Sidekick
State
Theories
Output
Query
Sledgehammer
Symbols
293,18 (9560/9875)
(isabelle,isabelle,UTF-8-Isabelle)
5:08 PM

Demo


Isabelle2018/First_Order_Terms - Example_Keyserver3.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Example_Keyserver3.thy (~/Repositories/LazyIntruderFormalization/)
298 subgoal by eval
299 subgoal by eval
300 subgoal by eval
301 subgoal by eval
302 subgoal by eval
Proof state
Auto update
Update
Search:
100%
∀l n. ∀𝒜∈reachable_constraints' keyserver.
#ℐ. (subst_domain ℐ = UNIV ∧ fv_subst_range ℐ = {}) ∧
spm.wf_trms (subst_range ℐ) ∧
spm.strand_sem_stateful {} {}
(unlabel (𝒜 @ [(l, send⟨Fun (Attack n) []⟩)])) ℐ
Documentation
Sidekick
State
Theories
Output
Query
Sledgehammer
Symbols
307,27 (9870/9875)
(isabelle,isabelle,UTF-8-Isabelle) I n m r o UG 966/1212MB 5:09 PM

Relative Soundness

- Our work is embedded into a whole framework for protocol verification in Isabelle
- nuFMC, like many other tools, assumes a typed model in which the intruder is restricted in what it can construct
- We have previously proven a typing result in Isabelle, namely that the restriction is sound for a large class of protocols: the type-flaw resistant protocols
- Thus, simply proving that the protocol is a member of this class lifts a typed-model proof of nuFMC to a proof for the untyped model as well
- We automatically check for type-flaw resistance
- In the future: automated checking of parallel composition conditions

Conclusion

We integrate an automatic protocol verification tool (nuFMC) with a proof assistant (Isabelle)

- Support for stateful protocols
- Completely automatic
  - Computing the fixed-point with nuFMC is automatic
  - The checks performed in Isabelle are also automatic
- Extremely high correctness guarantee
  - Isabelle verifies the output of nuFMC
- Is embedded into a whole framework for protocol verification in Isabelle
  - Allows us to automatically apply a typing result, lifting the security proofs from a typed model to an untyped one
  - It is possible to manually apply parallel compositionality results