flat-HLPSL

Proposal for a subprotocol and instantiation mechanism.

Idea: We want a

- parametrized/generic formulation of the protocol (independent of analysis considerations),
- be able to instantiate anything out to the last detail if we want.

Allow two types of role parameters:

- (...) Parameters that do not depend on the instantiation
- [...] Instances — Something that will be supplied from the "caller".

new-HLPSL: NSPK

```
role environment
  (Net: dolev_yao, S: agent, PK: keytable,
  sessions: (agent,Nat,(agent,public_key) list, (agent,Nat) list) list)

  role alice_or_bob_client (ID: identity, dummy: Nat,
     contacts: (agent,public_key) list
     sessions: agent list)

    role alice (A, B: agent, dummy: Nat)
      knowledge: A,PK(A),inv(PK(A)),S,PK(S),B,contacts
      Var: Na, fresh;
           Nb: unknown;
           state: initally 0;
      messages:
         state=0 /\ (B,PKB) in contacts
         --> state'=6 /\ Net_SND({A,Na}PKB)

         state=0 /\ (B,_) notin contacts
         --> state'=2 /\ Net_SND(A,B)
         ...
    elor alice
```

```
    role bob   (B: agent, dummy: Nat) ...

    glue:
      (/\_{(B,i).sessions} alice(ID,B,i)) /\ (/\_{i.Nat} bob(ID,i))
  elor alice_or_bob_client

  role server()
    knowledge: PK,inv(PK(S))
    messages:
       Net_RCV(A,B) --> Net_SND({B,PK(B)}inv(PK(S)))
  elor server

  glue:
    (/\_{(ID,i,contacts,ID_sessions).sessions}
        alice_or_bob_client(ID,i,contacts,ID_sessions))
    /\ server()
elor environment
```

new+old-HLPSL: NSPK

```
role environment
  (Net: dolev_yao, S: agent, PK: keytable,
  sessions: (agent,Nat,(agent,Nat) list) list)

  role alice_or_bob_client (ID: identity, dummy: Nat,
     sessions: agent list)

     role_Alice_Bob alice(A,B:agent,dummy:Nat)
                    bob(B:agent,dummy2:Nat)
                    server(S:agent,dummy3:Nat)
     var Na,Nb: nonce;
     knowledge A: A,B,PK(A),inv(PK(A)),PK(S);
               B: B,PK(B),inv(PK(B)),PK(S);
               S: S,PK,inv(PK(S));
     messages:
       1. A -> S: A,B
       2. S -> A: {B,PK(B)}(inv(PK(S)))
       3. A -> B: {Na,A}Kb
       4. B -> S: B,A
       5. S -> B: {A,PK(A)}(inv(PK(A)))
       6. B -> A: {Na,Nb}Ka
```

```
      7. A -> B: {Nb}Kb
    elore alice,bob,server

    glue:
      (/\_{(B,i).sessions} alice(ID,B,i)) /\ (/\_{i.Nat} bob(ID,i))
  elor alice_or_bob_client

  role server()
    knowledge: PK,inv(PK(S))
    messages:
       Net_RCV(A,B) --> Net_SND({B,PK(B)}inv(PK(S)))
  elor server

  glue:
    (/\_{(ID,i,ID_sessions).sessions}
        alice_or_bob_client(ID,i,ID_sessions))
    /\ server()
elor environment
```

The New IF

Sebastian Mödersheim

Information Security
ETH Zürich

The New IF

December State + Negated Facts

- IF with uninstantiated rules.
- Sections, rule parameters, type signatures.
- Syntactic beautIFications :-)
- Some modifications in property specifications (to be discussed).

Overview

- New IF Syntax and Semantics
  $\longrightarrow$ how can negation be handled with the lazy intruder?

- Examples:
  - NSPK with Keyserver
  - IKE (Frame)

- Example of a different intruder model
  $\longrightarrow$ how can step compression be achieved?

- Property specifications
  - Backwards LTL and state-based encoding
  - The "usual suspects": authentication and secrecy.

- Modelling Anonymity

New IF Syntax

Major points:

- Rules are not instantiated.
- The set of fact symbols is a parameter of the model.
  $\rightarrow$ Declare signature initially.
- Multiple initial states for automatic session generation.
- Negated Facts (to be commented below).
- $\mathrm{vars}(LHS) \supseteq \mathrm{vars}(RHS)$.
- Property specifications: later.
- Minor stuff: keytables etc.

Negation: Free Variables

$f_1, \ldots, f_n, \texttt{not}(g_1), \ldots, \texttt{not}(g_m) => h_1, \ldots, h_l$

- $\mathrm{vars}(f_i) \supseteq \mathrm{vars}(h_i)$ (Otherwise: new variables introduced by rule)
- $\mathrm{vars}(f_i) \supseteq \mathrm{vars}(g_i)$?
  . . . too restrictive! Example:

```
state([0,A,B,...]).not(knows_pk(A,B,PKB)) => ...
                                    ^^^
```

This case will occur and it is not a problem: PKB is used as a *wildcard* ("There is no value PKB such that A knows that PKB is the public key of B.")

Negation: Free Variables (2)

$f_1, \ldots, f_n,\ \texttt{not}(g_1), \ldots, \texttt{not}(g_m) => h_1, \ldots, h_l$

Arbitrary variables in g_i? For instance

```
not(f(M)).not(f(M)) => ...
```

Can this rule be applied in a state that contains a single fact `f(a)` ?

Negation: Free Variables (3)

$$f_1, \ldots, f_n, \texttt{not}(g_1), \ldots, \texttt{not}(g_m) => h_1, \ldots, h_l$$

Proposed solution: g_i may contain arbitrary variables, but they can only act as wildcards. Formally:

For all $v \in (\mathrm{vars}(g_i) \setminus vars(f_i))$
holds v occurs only once in the rule

$\rightarrow$ It could be replaced with an underscore.

Semantics of the New IF

- Basic definitions: ground terms, terms with pattern variables, substitutions for the pattern variables.

- *State*: Set (not multiset!) of terms.

- Transition relation:

$$
\begin{aligned}
&trans : Rule \times State \to \mathbb{P}(State)\\
&trans((f,g,h),S) = \{(S \setminus (f\sigma)) \cup (h\sigma) \mid\\
&\qquad \exists S_0 \subseteq S : \exists \sigma \in Gsub_{\mathrm{vars}(f)} :\\
&\qquad l\sigma = S_0 \wedge \textstyle\bigwedge_{g_0 \in g} \forall \sigma' \in Gsub_{\mathrm{vars}(g)\setminus\mathrm{vars}(f)} : g_0\sigma\sigma' \notin S\}\\
&Gsub_V \quad = \text{Ground substitutions with dom } V
\end{aligned}
$$

- Note that the following two rules are equivalent in this semantics:

```
not(f(M)).not(f(M2)) => ...
not(f(M))            => ...
```

How can the Lazy Intruder do it?

- As usual, we have to perform unification instead of matching, as both the state and the rules may contain variables.
- Try to find unifier σ for the positive facts with a subset of the current state.
- For every fact $\mathtt{not}(f_i(T))$:
  - Check if $f_i(T\sigma)$ is "literally" contained; if yes then fail.
  - Check if $f_i(T\sigma)$ can be unified under σ' with any fact of the current state; if yes, and

    $$\sigma' = [x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$$

    then add the constraint

    $$x_1 \neq t_1 \vee \ldots \vee x_n \neq t_n \; .$$

How can the Lazy Intruder do it? (2)

Handling of the inequality-constraints:

- Apply all substitutions also to the inequalities.
- Check if inequalities have become trivial ($7 \neq 5 \mapsto true$, $\texttt{crypt}(...) \neq Ka \mapsto true$, $7 \neq 7 \mapsto false$).
- Lemma: A simple constraint set with a conjunction of disjunctions of (non-trivial) inequalities is satisfiable, if the intruder can create fresh data arbitrarily: assign to every variable a different constant value.
- Lemma: None of the non-intruder variables can *creep* into the *from*-Constraints.

Overview

- New IF Syntax and Semantics
  $\longrightarrow$ how can negation be handled with the lazy intruder?

- Examples:
  - NSPK with Keyserver
  - IKE (Frame)

- Example of a different intruder model
  $\longrightarrow$ how can step compression be achieved?

- Property specifications
  - Backwards LTL and state-based encoding
  - The "usual suspects": authentication and secrecy.

- Modelling Anonymity

NSPK-Keyserver.nif

```
Initial state
(depending on instantiation):
  state([0,a,pk(a),pk(a)',s,pk(s),i]).
  state([3,b,pk(b),pk(b)',s,pk(s)]).
  knows_pk(b,a,pk(a)).
  state([s,pk,pk(s)'])

Step0a
  state([0,A,PKA,PKA',S,PKS,B]).
  not(knows_pk(A,B,PKB))
  =>
  state([2,A,PKA,PKA',S,PKS,B]).
  message(A,B)

Step0b
  state([0,A,PKA,PKA',S,PKS,B]).
  knows_pk(A,B,PKB)
  =>[Na]
  state([6,A,PKA,PKA',S,PKS,B,Na]).
  message({Na,A}PKB)
```

```
Step1 --- and Step 4
  state([S,PK,PKS']).
  message(A,B)
  =>
  state([S,PK,PKS']).
  message({B,PK(B)}PKS')

Step2
  state([2,A,PKA,PKA',S,PKS,B]).
  message({B,PKB}PKS')
  =>[Na]
  state([6,A,PKA,PKA',S,PKS,B,NA]).
  message({Na,A}PKB)
```

IKE

Individual protocol steps:

```
state([main_psk_0,A,Ka,...,B,PSKey]).
not(has_ISAKMP(A,B,DH_K,SKEYID_E,...))  % not(A knows ISAKMP for B).
=>[Ca]
message(Ca,ISAa)
state([main_psk_2,A,Ka,...,B,PSKey,Ca])

state([main_psk_6,A,Ka,...,B,PSKey,DH_x,G_DH_y]).
message(Ca,Cb,{B,<HASH_B>}(<SKEYID_E>))
=>
state([main_psk_end,A,Ka,...,B,PSKey,DH_x,G_DH_y]).
has_ISKAMP(A,B,{G_DH_y}DH_x,<SKEYID_E>,...)
```

IKE (Cont)

"Glue" Rules:

```
state([main_0,A,...]) => state([main_psk_0,A,...])
state([main_0,A,...]) => state([main_dsa_0,A,...])
...
state([main_0,A,...]) => state([agressive_psk_0,A,...])
...
state([main_psk_end,A,...]) => state([quick_no_pfs_0,A,...])
state([main_psk_end,A,...]) => state([quick_no_pfs_1,A,...])
```

These rules

- may depend on instantiation
- can be merged with those rules which describe the initial and the final steps of the sub-protocols.

Overview

- New IF Syntax and Semantics
  $\longrightarrow$ how can negation be handled with the lazy intruder?
- Examples:
  - NSPK with Keyserver
  - IKE (Frame)
- Example of a different intruder model
  $\longrightarrow$ how can step compression be achieved?
- Property specifications
  - Backwards LTL and state-based encoding
  - The "usual suspects": authentication and secrecy.
- Modelling Anonymity

A Different Intruder Model

In current model, `impersonate+eavesdrop` = `impersonate+divert`.
(Because of reachability/network model)

Different network model for "over the air" protocols (e.g. Bluetooth):

- All sent message of honest agents are received.
- The intruder can see, but not manipulate them.
- If the intruder has seen a message and sends a new message based on the learned knowledge, then his message is received after the one he has seen.

The Bluetooth intruder in IF

Idea:

- A queue of messages that are not yet received.
- The intruder may read any message in the queue.
- The intruder may add messages to the queue.
- IF realization: two counters $c1$ and $c2$:

```
state-rules:
 c1(T1).c2(T2).msgqu(T1,A,B,...).state([1,B,...]).not(c1(T2))
 =>
 c1(s(T1)).c2(s(T2)).msgqu(T2,B,A,...).state([3,B,...])
eavesdrop:
 msgqu(_,_,_,M) => msgqu(_,_,_,M).iknow(M)
forge:
 c2(T).iknow(M).iknow(A).iknow(B) => c2(s(T)).msgqu(T,A,B,M).iknow...
```

Step Compression for the New Intruder Model

It is tempting to argue as follows:

> Since the intruder can not divert messages, messages are guaranteed to be received. Then the receiver can directly (in the same transition) receive and react to the message, and so on.

Problem: this leads to a *chain reaction* (the entire protocol in one step), neglecting that after a message from A to B was sent and before B reacts, the intruder could send a message to A, posing to be B.

Solution: The different agent transitions can not be merged, but the intruder actions can be merged with the agent transitions.

Step Compression for the New Intruder Model (2)

- Eavesdrop can be merged into every agent transition: it is not a restriction that the intruder "automatically" sees every message.

- Forge: can not be merged, because not all messages need to originate from the intruder; but we can restrict the use of forge to messages that can be received by honest agents:
  - Compute the states of honest agents that will be reached after parsing all messages that are currently in the queue (and that will be received before any message the intruder can generate now).
  - To each of the honest agents that will exist then, the intruder can send a message from his current knowledge.

Overview

- New IF Syntax and Semantics
  $\longrightarrow$ how can negation be handled with the lazy intruder?
- Examples:
  - NSPK with Keyserver
  - IKE (Frame)
- Example of a different intruder model
  $\longrightarrow$ how can step compression be achieved?
- Property specifications
  - Backwards LTL and state-based encoding
  - The "usual suspects": authentication and secrecy.
- Modelling Anonymity

Property Specification

- Backwards-LTL is powerful enough to specify many important goals:

$$request(A) \Rightarrow \diamondminus witness(A)$$

- Formulated on *traces* of *events*.
- Can be encoded as state-based safety properties for IF.
- Optionally, tools can still work on the original specification
  ($\rightarrow$ events must be specified in rules).

Authentication

- Should be seen as a standard check: does a protocol provide authentication on all messages? (Or can the intruder manipulate anything, so that agents finish the protocol with a different interpretation of the protocol variables, or one agent finishes the protocol more often than the other.)

- Our current authenticate goals *correspond*[1] to Lowe's *strong agreement*.

- Weak authentication (e.g. Paulson's model): the same, if witness facts are never removed.

[1]. . . except for one subtle difference . . .

Authentication (2)

- Authentication is tricky thing, consider for instance Yahalom:
  - Lowe analyzed it with Casper/FDR and found no flaws.
  - Paulson verified it in Isabelle (typed, weak authentication goal).
  - OFMC has found a type-flaw attack that violated weak authentication.
  - OFMC (but Athena first) has found a (non-type-flaw) replay attack violating strong authentication.

Authentication (3)

- Problem with authentication on *static* (non-fresh) messages like agent names and long-term keys:
- Replay means that one participant has accepted a value more frequently than it was said by the other.
- This can not be checked by backwards-LTL unless we introduce counters.
- Simplification: allow only authentication on fresh messages (like nonces). Since they are *supposed to be* fresh, it is a violation of authentication, if an agent accepts the same fresh message two times.
- This can be checked in backwards-LTL.
- I would like to also allow authentication on non-atomic messages (that however must contain something fresh!) — like Diffie-Hellman (half-) keys.

Authentication (4)

- `A weakly authenticates B [on M]`

- Requirement: M can be derived both by A and by B at the latest when A receives the last message of the protocol and M contains something fresh.

- If the optional message is not given, by default all protocol submessages that satisfy the above requirement are authenticated.

- Intuition: when an agent a in role A finishes the protocol, then there is an agent in role B that agrees with a on the variables A, B, and M.

- Let respective events $witness(A, B, varM, M)$ and $request(A, B, varM, M)$ be defined:

$$\Box(request(A) \rightarrow \diamondsuit witness(A))$$

Authentication (5)

- `A weakly authenticates B [on M]`

- State-Based encoding:

```
Rule of witness:
   ... => ... . witness(...)

Rules of request:
  state(...).message(...).witness(...) => state(...).witness(...)..
  state(...).message(...).not(witness(...)) => error() ...

Goal:
  error()
```

Authentication (6)

- `A authenticates B [on M]`
- Requirement: M can be derived both by A and by B ...
- Intuition: when an agent a in role A finishes the protocol, then an agent in role B exists that agrees with a on the variables A, B, and M, and a has never finished the protocol with the same value for M before.
- $\Box(request(A) \rightarrow ((\diamondminus witness(A)) \wedge (\ominus \boxminus \neg(request(A)))))$
- State-Based encoding:

```
Rule of witness:   ... => ... . witness(...)
Rules of request: state(...).message(...).witness(...) => state(...)...
                  state(...).message(...).not(witness(...)) => error() ...
Goal:             error()
```

Secrecy

- The tricky thing here is not the temporal structure, but only the question who may know what.
- Current problem: based on session numbers which can be fooled;
  better: based on agent names.
- Fresh secret: when it is created, it is declared for whom it is intended by facts of the form

  `... => ... . secret(M,A).secret(M.B)`
- The goal will be that the intruder knows something that is a secret and the intruder does not officially share the secret:

  `secret(M,A).iknow(M).not(secret(M,i))`

Overview

- New IF Syntax and Semantics
  $\longrightarrow$ how can negation be handled with the lazy intruder?
- Examples:
  - NSPK with Keyserver
  - IKE (Frame)
- Example of a different intruder model
  $\longrightarrow$ how can step compression be achieved?
- Property specifications
  - Backwards LTL and state-based encoding
  - The "usual suspects": authentication and secrecy.
- Modelling Anonymity

Anonymity — Example IKE

1. $A \to B: \quad C_A, ISA_A$
2. $B \to A: \quad C_A, C_B, ISA_B$
3. $A \to B: \quad C_A, C_B, g^x, N_A$
4. $B \to A: \quad C_A, C_B, g^y, N_B$
5. $A \to B: \quad C_A, C_B, \{ID_A, SIG_A\}_{SKEYID_e}$
6. $B \to A: \quad C_A, C_B, \{ID_B, SIG_B\}_{SKEYID_e}$

- ID_A and ID_B represent the real identities of the agents (while the adresses used for communication between A and B may be temporary addresses — these temporary adresses are never mentioned in the messages). The real identities are only transmitted in the encrypted messages 5 and 6.
- Cathy Meadows reports an attack that is based on binding the keys to the temporary names A and B — this is a misunderstanding:
- The negotiated keys are bound to the identities ID_A and ID_B, hence the goal should be: ID_A `authenticate` ID_B `on` $SKEYID, SKEYID_a, \ldots$
- Anonymity is a harder goal than simply the secrecy of ID_A and ID_B: secrecy of the real identities can be trivially violated by the agent starts a session to the intruder or responds to an intruder-initiated communication.

A Solution for the Identity Problem

- A and B represent temporary addresses, however they play no role in the protocol.
- Everything should be associated with ID_A and ID_B, representing the real identities.
- Idea: let A and B represent the real identities:
  5. $A \rightarrow B : \quad C_A, C_B, \{A, SIG_A\}_{SKEYID_e}$
  6. $B \rightarrow A : \quad C_A, C_B, \{B, SIG_B\}_{SKEYID_e}$
- If temporary addresses are explicitly used in the protocol they can be nonces (that might need to be authenticated).
- However, in our protocol model, the real identities are now revealed with every message:

  `m(1,a,a,b,msg1,2)`

  $\Rightarrow$ The intruder knows both a and b and the session number 2, though the message doesn't reveal this information.
- Suggestion: assume that only the information explicitly mentioned in the protocol description is transmitted!
- The technical sender/receiver information (like dynamic IP adresses) typically doesn't play a role in these protocols. Whenever it does, then one can explicitly model it using nonces!

Conclusions

- Powerful formalism for specifying transition systems.
- Negation can be added and it increases expressiveness,
- ... but one must be carefull e.g. with semantics and implementation.
- It seems possible to encode protocols with "non-standard" control, data structures, intruder model, and goal strategy.