Design of a CIL Connector to SPIN

Li Yongjian, Xuerui

Institute for Software, Chinese Academy of Sciences

Beijing, China[1]

The State Key Laboratory of Information Security[2]

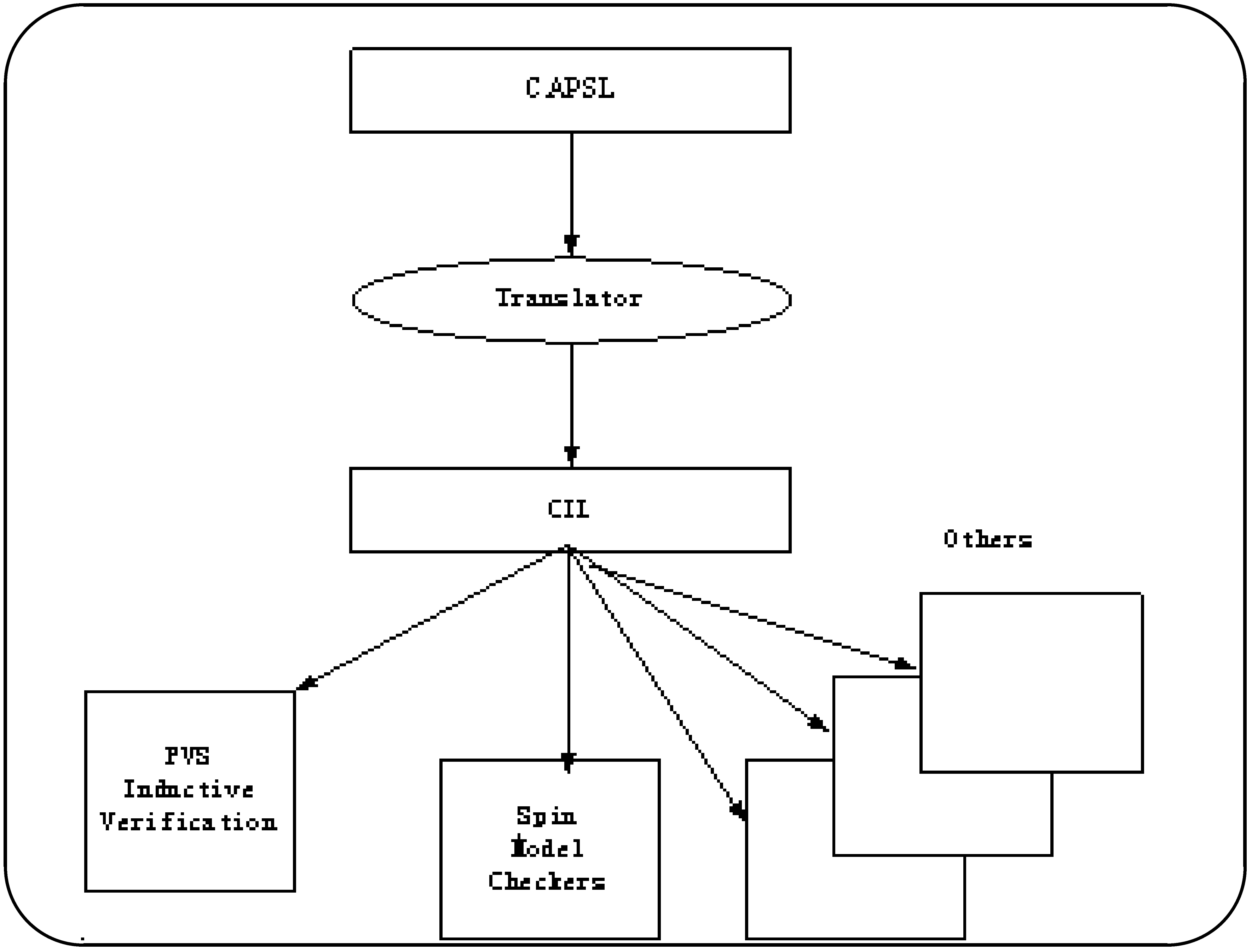
lyj238@ios.ac.cn

Outline

1. CAPSL
2. CIL
3. Spin
4. CIL-Spin Connector
5. Related work and Conclusion

CAPSL

1. Motivation of CAPSL (Common authentication Protoctocl specification Language)
   - a natural style for users to specify cryptographical protocols
   - just message list style in textbooks
   - CAPSL Intermediate language (CIL) as an interface to various formal tools
   - CAPSL spec file $\xrightarrow{Translator}$ CIL file $\xrightarrow{CIL-X \text{ connector}}$ X-tool format
   - users need construct CIL-X connector by the X tool
2. Architecture of CAPSL environment


CAPSL
Translator
CIL
Others
PVS
Inductive
Verification
Spin
Model
Checkers

3. Syntax of CAPSL (an example of NSPK protocol CAPSL file)
   - Type specification
   - Protocol specification
   - environment specification
   - An example–NSPK CAPSL file

```
TYPES PKUser: Principal;
FUNCTIONS
pk(PKUser): Pkey;
sk(PKUser): Pkey, PRIVATE,
                CRYPTO;
VARIABLES
X: Field;
U: PKUser;
AXIOMS
{{X}pk(U)}sk(U) = X;
{{X}sk(U)}pk(U) = X;
INVERT {X}pk(U): X | sk(U);
INVERT {X}sk(U): X | pk(U);
END;
```

```
PROTOCOL NSPK;
 VARIABLES
A, B: PKUser;
 Na, Nb: Nonce, CRYPTO;
ASSUMPTIONS
 HOLDS A: B;
MESSAGES
A   -> B: {Na, A}pk(B);
B   -> A: {Na, Nb}pk(A);
A   -> B: {Nb}pk(B);
GOALS
SECRET   Nb: B, A;
 PRECEDES B : A | Na
END;
```

```
ENVIRONMENT Test1;
 IMPORTS  NSPK;
 CONSTANTS
  Alice, Bob: PKUser;
    Intruder: PKUser, EXPOSED;
    S12Nb£″Nonce, Exposed;
    S12Na, S13Nb: Nonce;
 AGENT   A1 HOLDS
  A = Alice;       Na=A1Na;
     B = Intruder;
 Agent B1 HOLDS
  B    = Intruder; Nb=B1Nb;
  AGENT B2 HOLDS
     B = Bob;      Nb=B2Nb;
  ORDER (A1||B1||B2) ;
END;
```

– needed in model checking
– setting up a test environment of the protocol

CIL

1. Syntax
   - Symbol table
   - Slot table
   - Axioms
   - rewrite rules
   - goals
   - environments
   - NSPK cil

```
CILspec(
symbols(
symbol(Object,type,ids(),Object,props()),
symbol(BASIC,op,ids(),Tspec,props()),
symbol(Role,type,ids(),Object,props()),
```

```
........
symbol(endprelude,op,ids(),Boolean,props()),
symbol(NSPK,op,ids(),Pspec,props()),
symbol(A,pvar,ids(),PKUser,props()),
symbol(B,pvar,ids(),PKUser,props()),
symbol(Na,pvar,ids(),Nonce,props(CRYPTO,FRESH)),
symbol(Nb,pvar,ids(),Nonce,props(CRYPTO,FRESH)),
symbol(Test1,op,ids(),Espec,props()),
symbol(Alice,op,ids(),PKUser,props()),
symbol(Bob,op,ids(),PKUser,props()),
symbol(Intruder,op,ids(),PKUser,props()),
symbol(B1Nb,op,ids(),Nonce,props()),
.....
symbol(UNK,pvar,ids(),Principal,props())
),
slots(
```

```
slot(A,roleA,1),
slot(B,roleA,2),
....
slot(Nb,roleA,4)
),
axioms(
eqn(first(cat(Al,Xl)),Al),
eqn(rest(cat(Al,Xl)),Xl),
invertible(cat(Al,Xl),Al,terms()),
invertible(cat(Al,Xl),Xl,terms()),
eqn(sd(Kl,se(Kl,Xl)),Xl),
eqn(se(Kl,sd(Kl,Xl)),Xl),
....
invertible(con(Xl,Yl),Xl,terms()),
invertible(con(Xl,Yl),Yl,terms()),
```

```
eqn(endprelude,true)
),
assums(loc(nodes(node(roleA,0),node(roleB,0)),holds(A,ids(B)))),
rules(
rule(facts(),ids(),facts(state(roleA,0,terms(A,B)))),
rule(facts(),ids(),facts(state(roleB,0,terms(B)))),
rule(
facts(state(roleA,0,terms(A,B))),
ids(Na),
facts(state(roleA,1,terms(A,B,Na)),msg(A,B,terms(ped(pk(B),cat(Na,A)))))
),
rule(
facts(state(roleB,0,terms(B)),msg(UNK,B,terms(ped(pk(B),cat(Na,A))))),
ids(Nb),
facts(
```

```
state(roleB,2,terms(B,Na,A,Nb)),
msg(B,A,terms(ped(pk(A),cat(Na,Nb))))
)
),
rule(
facts(
state(roleA,1,terms(A,B,Na)),
msg(UNK,A,terms(ped(pk(A),cat(Na,Nb))))
),
ids(),
facts(state(roleA,3,terms(A,B,Na,Nb)),msg(A,B,terms(ped(pk(B),Nb))))
),
rule(
facts(state(roleB,2,terms(B,Na,A,Nb)),msg(UNK,B,terms(ped(pk(B),Nb)))),
ids(),
```

```
facts(state(roleB,3,terms(B,Na,A,Nb)))
)
),
goals(
loc(nodes(node(roleA,3),node(roleB,3)),secret(Nb,ids(B,A)))
),
envs(
environment(
Test1,
agents(
agent(A1,eqns(eqn(A,Alice),eqn(Na,A1Na),eqn(B,Intruder))),
agent(B1,eqns(eqn(B,Intruder),eqn(Nb,B1Nb))),
agent(B2,eqns(eqn(B,Bob),eqn(Nb,B2Nb)))
),
exposed(terms(Intruder,B1Nb)),
```

order(par(par(A1,B1),B2))
)
)
)

2. Semantics
   - MSR (multiset rewriting)
   - states —multiset of facts
   - state transition rules– left facts removed, right facts added

Spin

- powerful general purpose model checker
- popular in verification of software and communication protocols.
- but few successful application in verification of cryptographical protocols.

- our motivation to implement CIL-Spin connectors
  - lack of free analyzing tools forced us to produce our own tools
  - Spin is free and efficeient
  - helps us understand CIL and CAPSL
  - helps us understand modecl-checking techniques put in proper place

CIL-Spin connector

- compute facts used in test environment
  - only considering facts involved possibly used by the intruder to launch attacks
  - compute them by the message formats, agents assignments in environment specification
- Data structure to encode facts
  - recursively defined, messgaes having

structures

- A Fact array to encode the binary tree of the structure of all facts
- type definition
- A fragment of the fact array typedef

```
TFact{
        mtype oper;
        int operand1;
        int operand2;}
      TFact Facts[total_fact]
```

- communication channel chan comm = [0] of mtype, short;

CIL-Spin connector

- honest principals
  - formal parameters
  - local variable
  - statements
    - message exchange
    - checking messages received correct or not
    - special signal events to express beliefs, for goals verification

- A sample initiator code proctype

roleA(mtype Na;mtype A;mtype B)

```
{ short Msgno; TFact Msg;
mtype Loc_Na;mtype Loc_A;mtype
Loc_B;
mtype Loc_Nb; mtype Memory_Nb;
atomic{
cons_Msg0(B,Na,A,Msg,Msgno);
comm!Msg0(Msgno); };
atomic{
```

```
comm?Msg1(Msgno);
destruct_Msg1(Loc_A,Loc_Na,Loc_Nb,Msg,Msgno);
Memory_Nb=Loc_Nb;
A==Loc_A;
Na==Loc_Na;} ;
atomic{
cons_Msg2(B,Memory_Nb,Msg,Msgno);
comm!Msg2(Msgno); }
}
```

- what does the connector do

- allocating local variables
- allocating formal parameters
- automatically generate messages exchanging statements
- including message encoding before sending,
- message decoding after receiving the message

Spy Model

- the presentation of knowledge of the spy

  spy_known[total_fact]

- deduction rule—

  <Conclusion, Assumptions>

  <cat(NaS12,Alice),{NaS12}>∈SqDeductions

  < NaS12,{cat(NaS12,Alice}>∈

  UnSqDeductions

  < pke(pk(Intruder), NbS13),{ NbS13}

$> \in$ EncryptionDeductions

$<$ NbS13,{pke(pk(Intruder), NbS13)}

$> \in$DecryptionDeductions

- type definition

```
typedef TDeduction{
bit Deduced;
int Conclusion;
int Assumption1,Assumption2;
};
TDeduction Deductions[DeductionNum];
```

- behaviors of the spy
  - Overhear and/or intercept any message
  - Infer new facts by rules and knowledge
  - Fake messages
- the promela code proctype PI ()

```
{ TFact M;
short Mno,headAssumption;
short A_q_head=0, A_q_tail=0;
short D_q_tail=0, index =0;
TDeduction CurDeduction;
```

```
Assumptionqueue[total_fact];
do
:: atomic { comm?Msg0(Mno);
addKnowledge(M,Mno);}
::atomic { sel_Msg(Msg0,Mno);
comm!Msg0(Mno);}
::atomic { comm?Msg1(Mno);
addKnowledge(M,Mno);}
::atomic { sel_Msg(Msg1,Mno);
comm!Msg1(Mno);}
```

```
::atomic { comm?Msg2(Mno);
addKnowledge(M,Mno);}
::atomic { sel_Msg(Msg2,Mno);
comm!Msg2(Mno);}
od;}
```

- the inference system of the spy
  - a queue to store facts just infered
  - combined with deduction rules, facts will be used to infer new facts again
  - this procedure will not stop until no new

facts infered

– abstract algorithm Fuction addKnowledge(Fact M)

{Assumption_Queue:=null;

put(Assumption_Queue, M);Spy_Known:= Spy_Known$\cup${M};

repeat

Head_Fact:=get(Assumption_Queue);/*get the head out of Assumption_Queue and assign it to Head_Fact*/

Deduces_{HF}:={<

Conclusion,Assumptions> |

<

Conclusion,Assumptions>∈Unused_Deductions,

Conclusion∉Spy_Known,

Head_Fact∈ Assumptions,

Assumptions⊆ Spy_Known };

Unused_Deductions :=

Unused_Deductions- Deduces_{HF};

for all <Conclusion, Assumptions>∈

Deduces$_{HF}$ {

put(Assumption_Queue,

Conclusion);Spy_Known:= Spy_Known $\cup$

{Conclusion};}

until (Assumption_Queue<>null)

}

Setting up the test environment

- initialization of facts array, deduction array, intial knowledge of the Spy
- setting up all the honest agents by the parameters defined in environment specification

Expressing security Goal

- insert some additional signal events to indicate the agent's beliefs.
- use LTL to define security property

secrecy property

- $secret(Nb, ids(B, A)))$—if honest principals a, b acting roles $roleA$, $roleB$ finished a session, then the actual value of the protocol nonce variable Nb should not be detected by the intruder provided that both a and b are honest.
- $RoleA\_Claim\_secret\_Nb[totol\_inst\_of\_RoleB]$

allocated to store values of the protocol variable Nb claimed by the honest principals acting the role RoleB

- insert secrecy_claiming codes into the body of proctype RoleB
- LTL formalism :

[](!(RoleB_Claim_secret_Nb[0]==-1)->Spy_Known[RoleB_Claim_secret_Nb[0]]==0)&&

. . . . . . &&

!(RoleB_Claim_secret_Nb[totol_inst_of_RoleB-

1]==-1)->

Spy_Known[RoleB_Claim_secret_Nb[totol_inst_of_RoleB-

1]]==0))

Authentication Goal

- $precedes(B, A, ids(Na)))$ – –if an honest principal b acting the role roleB finished a session, then there exists an honest principal a acting role roleA helds the same values as b on the protocol variables A, B, and Na.
- $Commit\_on\_Na[totol\_inst\_of\_RoleB, totol\_inst\_of\_RoleA]$ to store all the encoding number of the possible instances of the protocol variable

Na which are commited by b acting the role $RoleB$

- $Init\_on\_Na[totol\_inst\_of\_RoleA, totol\_inst\_of\_RoleB]$ to store all the encoding number of the possible instances of the protocol variable Na origninated by a acting the role $RoleA$
- insert Na_committing codes into the body of proctype RoleB
- insert Na_Iniitializing codes into the body of

proctype RoleA

- LTL formalism :

  []((!(Commit_on_Na[0, 0] ==-1)->

  Commit_on_Na 0, 0]== Init_on_Na[0, 0])&&

  ....

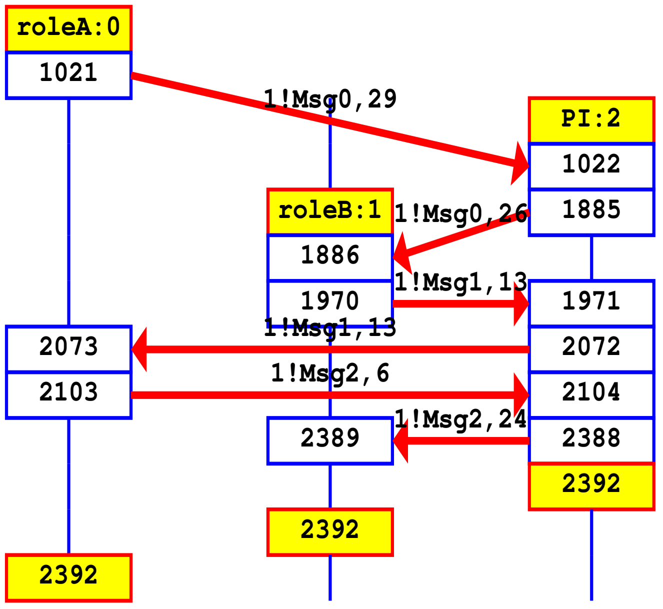
  !(Commit_on_Na[i, j]=-1)-> Commit_on_Na

  [i,j]== Init_on_Na[j, i]) &&

  ....

)

- sample attacks


roleA:0
1021
1!Msg0,29
PI:2
1022
1885
roleB:1
1!Msg0,26
1886
1!Msg1,13
1970
1971
1!Msg1,13
2073
2072
1!Msg2,6
2103
2104
1!Msg2,24
2389
2388
2392
2392
2392

Conclusion

- A general technique to encode all the facts involved.
- A general approach to model all honest principals.
- A general intruder model.
- A CIL-Spin connector.
- efficient enough for find attacks
- better visualization on attack graphs

- Limitation
  - a finite system
  - protocol goal is subtle
  - can not prove protocol correct