

## *A Cryptographic Protocol Example*

We can illustrate the power of this model checking technique for safety properties of infinite state systems by showing how it can be used to find subtle **attacks** for **cryptographic protocols**, including some that have been used extensively and have been considered secure for a long time.

One such protocol is the 1978 Needham-Schroeder authentication protocol (NSPK) for which a subtle “man-in-the-middle” attack was found by G. Lowe in 1996 using model checking.

The goal of the NSPK Protocol is to provide **authentication** of two agents who want to be assured of each other’s identity before they exchange safety-critical data.

## *A Cryptographic Protocol Example (II)*

That is, an intruder should not be allowed to impersonate another agent. For this purpose, initiator and responder of a communication mutually authenticate each other.

NSPK uses **public key cryptography**, i.e., each agent has a public key which can be accessed by all agents, and a secret key which is the inverse of the public key.

Moreover, **nonces** are used in the protocol. Nonces are freshly generated, unguessable random numbers to be used in a single run of the protocol.

## *A Cryptographic Protocol Example (III)*

Here is a textbook-style simplified description of NSPK:

Message 1       $A \rightarrow B : A.B.\{N_a, A\}_{PK(B)}$

Message 2       $B \rightarrow A : B.A.\{N_a.N_b\}_{PK(A)}$

Message 3       $A \rightarrow B : A.B.\{N_b\}_{PK(B)}$

This level of description is **ambiguous**, in that a fair amount of **implicit assumptions** are **left unspecified**. An object-oriented rewriting logic specification of the protocol (developed in joint work with G. Denker and C. Talcott) makes these assumptions explicit, and allows model checking.

## *Maude Specification of NSPK*

We first specify key **algebraic properties** of the **cryptographic infrastructure** in a functional module.

```
fmod DATATYPES is
  sorts Key Field Nonce Principal Run Role EstabComm .
  subsort Nonce Principal Key < Field .
  op keypair : Key Key -> Bool [comm] .
  op ped : Key Field -> Field . *** encryption function
  op n : Principal Nat -> Nonce .
  ceq ped(sk,ped(pk,f)) = f  if keypair(sk,pk) .
  ...
endfm
```

The protocol itself, as well as the actions of an attacker, are specified as follows (fragment):

## *Maude Specification of NSPK (II)*

```
class Agent | e_com: EstabCom, sec_key: Key, role_i: Run, role_r: Run,  
             d_com: FieldSet cnt: Nat .  
  
msg from_to_send_ : Principal Principal Field -> Message .  
  
vars A B P : Principal . vars RI RR : Run . vars NI : Nonce . ...  
rl [BeginRun] :  
  < A : Agent | role_i: RI, d_com: B U S, cnt: J >  
  => < A : Agent | role_i: RI U (n(A,J),B,mtfield), d_com: S, cnt: J + 1 >  
    from(A)to(B)send(ped(pk(B),n(A,J),A)) .  
  
crl [Message1Rec] :  
  < B : Agent | sec_key: SKB, role_i: RI, role_r: RR, cnt: J >  
  from(A)to(B)send(ped(PKB,F,A))  
  => < B : Agent | role_r: RR U (n(B,J),A,F), cnt: J + 1 >  
    from(B)to(A)send(ped(pk(A),F.n(B,J)))  
  if keypair(SKB,PKB) and not(F in RR) .
```

## *Maude Specification of NSPK (III)*

```
crl [Message2RecCorrect] :  
  < A : Agent | sec_key: SKA, role_i: RI U (NI,P,mtfield), e_com: C >  
  from(B)to(A)send(ped(PKA,F))  
=> < A : Agent | role_i: RI, ECom: C U (i,NI,B,rest(F)) >  
    from(A)to(B)send(ped(pk(B),rest(F)))  
  if keypair(SKA,PKA) and (B == P) and (NI == first(F)) .  
  
crl [Message2RecIncorrect] :  
  < A : Agent | sec_key: SKA, role_i: RI U (NI,P,mtfield) >  
  from(B)to(A)send(ped(PKA,F))  
=> < A : Agent | role_i: RI >  
  if keypair(SKA, PKA) and (NI == first(F)) and (B /= P) .
```

## *Maude Specification of the Intruder*

```
class Intruder | e_com: EstabCom sec_key: Key ncs: FieldSet,  
                msgs: FieldSet agents: FieldSet role_i: Run,  
                role_r: Run d_com: Field cnt: Nat.
```

```
cr1 [IntruderFakeMessage] :  
  < I : Intruder | ncs: N U F, agents: S U A U B >  
  => < I : Intruder | ncs: N U F > from(A)to(B)send(ped(pk(B),F))  
  if B /= I .
```

```
similar: IntruderInterceptMessage, IntruderOverhearMessage,  
         IntruderReplayMessage
```

## *The State Predicate of an Attack*

In a topmost version of the specification, the situation where **authentication information has been compromised** is specified by the following state predicate:

```
op attack? : Configuration -> Prop .
```

```
ceq attack?(boundary(  
  < INTR : Intruder | ecom : EC, rolei : RI, roler : RR,  
    ncs : fset+(fset+(FSET1, N1), N2) >  
  < A : NSPKAgent | ecom : ecom+(EC2, ecom(ROLE,N1,B,N2)) >  
  Conf)) = true  
if (not(inEstabCom(ecom(r,N2,A,N1),EC))  
  and not(inEstabCom(ecom(i,N2,A,N1), EC))  
  and not(in(N1,RI)) and not(in(N1,RR))  
  and not(in(N2,RI)) and not(in(N2,RR))  
  and B /= INTR) == true .
```



## *Finding an Attack*

The relevant **safety property** is that no such attack is possible under reasonable initial conditions. For example, we can consider a simple scenario with two agents and an attacker given by an initial state `cf2Agents1Intruder` equationally defined in the obvious manner. Then the desired safety property is  $P \rightarrow \Box Q$  with:

$$P = (S = \text{cf2Agents1Intruder})$$

$$Q = \neg(\text{attack?}(S) = \text{true}).$$

Maude's `search` command finds Lowe's countarexample to such a property:

## *Finding an Attack (II)*

```
Maude> search [1] cf2Agents1Intruder =>+ C:Configuration s.t.
                                     attack?(C:Configuration) = true .
search [1] in NSPK : cf2Agents1Intruder =>+ C:Configuration such that
                                     attack?(C:Configuration) = true .

Solution 1 (state 37826)
states: 37827 in 25350ms cpu (44300ms real)
C:Configuration -->
  boundary(< alice : NSPKAgent | cnt : 2,dcom : mtfset,roler
    : mtrun,rolei : mtrun,seckey : skalice,ecom : ecom(i, n(alice, 1), mrx, n(
    bob, 1)) > < bob : NSPKAgent | cnt : 2,dcom : mtfield,roler : mtrun,rolei :
    mtrun,seckey : skbob,ecom : ecom(r, n(bob, 1), alice, n(alice, 1)) > < mrx
    : Intruder | cnt : 1,dcom : mtfield,roler : mtrun,rolei : mtrun,seckey :
    skmrx,ecom : mtecom,agents : fset+(alice, bob, mrx),ncs : fset+(mtfset, n(
    alice, 1), n(bob, 1)),msgs : fset+(mtfset, ped(pkalice, cat(n(alice, 1), n(
    bob, 1)))) >)
```

### *Finding an Attack (III)*

We can find the actual sequence of rewrites leading to the attack by giving to Maude the command `show path 37826` . A detailed trace is then shown, corresponding to the sequence of rewrite rule applications:

```
[BeginRun] ; [IntruderAcceptEveryMessage1] ; [IntruderFakeMessage1] ;
```

```
[Message1Rec] ; [IntruderInterceptMessage2] ; [IntruderReplayMessage] ;
```

```
[Message2Rec] ; [IntruderAcceptEveryMessage3] ;
```

```
[IntruderFakeMessage3] ; [Message3Rec]
```