Master M2 Cybersecurity January, 2024
Course : Cryptographic Engineering Duration 50 minutes

Problem

Your goal is to complete a model of the Needham-Schroeder-Lowe protocol in Tamarin (its Alice-Bob notation is given below).

```
     1. I -> R: {'1',ni,I}pk(R)
     2. I <- R: {'2',ni,nr,R}pk(I)
     3. I -> R: {'3',nr}pk(R)
where:   * pk(X) - long-term public key of X
         * ni - nonce generated by I (once per session)
         * nr - nonce generated by R (once per session)
         * the goal is to exchange the nonces ni and nr as fresh shared secrets
```

In this protocol, each party `A` has a long-term private key `ltkA` and a corresponding public key `pk(A) = Pk(A, ltkA)`. An asymmetric encryption scheme is used, and `{m}pk(A)` denotes the encryption of message `m` with the public key of `A`.

To start a session, the initiator `I` first creates a fresh nonce `ni`. He then concatenates `ni` with its `I`'s identity, encrypts the result using the public key of `R`, and sends it to `R`. The responder `R` stores the received value, generates a fresh nonce `nr` , concatenates `ni` with `nr` and its own identity, encrypts the result using the public key of `I`, and sends it to `I`. Finally, the initiator sends back to the responder the encryption of `nr`. After a completed session, both parties assume that the nonces `nr` and `nr` are fresh and secret. The incomplete modelization you should consider is given in Figures 1 and 2. Exercises:

1. Can you formulate an **"executable"** lemma in Tamarin that says that both participants (an initiator and a responder) can complete a session (even if the adversary does not corrupt anyone)?

   Hint: You should use an"exists-trace" assertion that uses the predicates OUT...(...), IN...(...) and RevLtk(...) .

2. Can you formulate a **"secrecy_claim"** lemma in Tamarin that models the secrecy property of the generated nonces (with respect to a session where both participants are not corrupted) ?

   Hint: The assertion should use the predicates Secret(...), K(...) and RevLtk(...).

3. We recall the definition of (perfect) forward secrecy: compromise of long-term keys of a set of principals does not compromise the session secrets established in previous protocol runs involving those principals. Can you formulate a **"PFS_secrecy_claim"** lemma that models the PFS secrecy property of the session secrets?

   Hint: You should complete the previous lemma by allowing the adversary to corrupt the involved participants after the completion of the target session .

4. Can you complete the rules of the protocol and formulate a **"PFS_secrecy_claim_I"** lemma that models the PFS secrecy property of the session secrets from the point of view of the initiator?

   Hint: You should add an action fact SecretI(...) on one of the rules corresponding to the side of the initiator.

5. Complete the rules of the protocol such that the lemma **"injective_agree"** should correspond to the Injective agreement property from the perspective of both the initiator and the responder.

   Hint: You should add action facts Commit(...) and Running(...) on the right rules.

```
theory NSLPK
begin
builtins: asymmetric-encryption
/*.   Protocol:    The  Needham-Schroeder-Lowe Public Key Protocol  */

// Public key infrastructure
rule Register_pk:
  [ Fr(~ltkA) ]  --> [ !Ltk($A, ~ltkA), !Pk($A, pk(~ltkA)), Out(pk(~ltkA)) ]

rule Reveal_ltk:
  [ !Ltk(A, ltkA) ] --[ RevLtk(A)    ]-> [ Out(ltkA) ]

/* We formalize the following protocol
  protocol NSLPK
    1. I -> R: {'1',ni,I}pk(R)
    2. I <- R: {'2',ni,nr,R}pk(I)
    3. I -> R: {'3',nr}pk(R)
    */

rule I_1:
  let m1 = aenc{'1', ~ni, $I}pkR in
    [ Fr(~ni) , !Pk($R, pkR) ] --[ OUT_I_1($I, $R, ~ni) ]->
    [ Out( m1 ) , St_I_1($I, $R, ~ni) ]

rule R_1:
  let m1 = aenc{'1', ni, I}pk(ltkR) m2 = aenc{'2', ni, ~nr, $R}pkI in
    [ !Ltk($R, ltkR) , In( m1 ) , !Pk(I, pkI) , Fr(~nr) ]
  --[ IN_R_1( $R, I, ni ) , OUT_R_1( $R, I, ni, ~nr )  , Running(I, $R, <'init',ni,~nr>) ]->
    [ Out( m2 ) , St_R_1($R, I, ni, ~nr) ]

rule I_2:
  let m2 = aenc{'2', ni, nr, R}pk(ltkI)  m3 = aenc{'3', nr}pkR in
    [ St_I_1(I, R, ni) , !Ltk(I, ltkI) , In( m2 ) , !Pk(R, pkR) ]
  --[ IN_I_2( I, R, ni, nr), OUT_I_2(I, R, nr),  Commit (I, R, <'init',ni,nr>) ,
  Running(R, I, <'resp',ni,nr>), Secret(I,R,nr), Secret(I,R,ni), SecretI(I,R,nr), SecretI(I,R,ni)]->
    [ Out( m3 )  ]

rule R_2:
  let m3 = aenc{'3', nr}pk(ltkR) in
   [ St_R_1(R, I, ni, nr) , !Ltk(R, ltkR) , In( m3 ) ]
  --[ IN_R_2( R, I, nr ) ,  Commit (R, I, <'resp',ni,nr>), Secret(R, I, ni), Secret(R, I, nr) ]->
   [ ]
```

Figure 1: A Model of NSLPK

```
/* 1.  */
lemma executable:
 exists-trace
"Ex I R ni nr #i1 #i2 #i3 #i4 .
                      OUT_I_1(I, R, ni)@#i1 & IN_R_1(R, I, ni)@#i2 &
                      OUT_R_1(R, I, ni, nr)@#i2 & IN_I_2(I, R, ni, nr)@#i3 &
                      OUT_I_2(I, R, nr)@#i3 & IN_R_2(R, I, nr)@#i4
                       & not (Ex A #r. RevLtk(A) @ r) "

 /*2.   If there exists a  session whose one secret s is known to the Adversary,
    then the long term key of one of the two participants involved in that session was revealed */
lemma secrecy_claim:
" All A B s #i.   Secret(A, B, s) @ i ==>
       ( (not(Ex #j. K(s) @ j)) | ( Ex #r. RevLtk(A) @ r) | (Ex #r. RevLtk(B) @ r) )
"

 /*3.  If there exists a  session whose one secret s is known to the Adversary, then the long term key
 of one of the two participants involved in that session was revealed before that session  */
lemma PFS_secrecy_claim:
 " All A B s #i.   Secret(A, B, s) @ i ==>
       ( (not(Ex #j. K(s) @ j)) | ( Ex #r. RevLtk(A) @ r & r<i ) | (Ex #r. RevLtk(B) @ r & r<i ) )
"

 /*4.  If there exists a session where the initiator has accepted a secret s and s is known to the
Adversary, then the long term key of one of the two participants involved in that session
was revealed before that session */
lemma PFS_secrecy_claim_I:
" All A B s #i.   SecretI(A, B, s) @ i ==>
       ( (not(Ex #j. K(s) @ j)) | ( Ex #r. RevLtk(A) @ r & r<i ) | (Ex #r. RevLtk(B) @ r & r<i ) )
"

 //5. Injective agreement from the perspective of both the initiator and the responder.
lemma injective_agree:
  " /* Whenever somebody commits to running a session, then*/
    All actor peer params #i.   Commit(actor, peer, params) @ i
      ==>
        /* there is somebody running a session with the same parameters */
          (Ex #j. Running(actor, peer, params) @ j & j < i
            /* and there is no other commit on the same parameters */
            & not(Ex actor2 peer2 #i2.  Commit(actor2, peer2, params) @ i2 & not(#i = #i2))
          )
        /* or the adversary perform a long-term key reveal on actor or peer */
        | (Ex #r. RevLtk(actor) @ r)     | (Ex #r. RevLtk(peer)  @ r)
  "

end
```

Figure 2: Properties