Verification of NSLPK and Some Tips for Construction of Proof Score

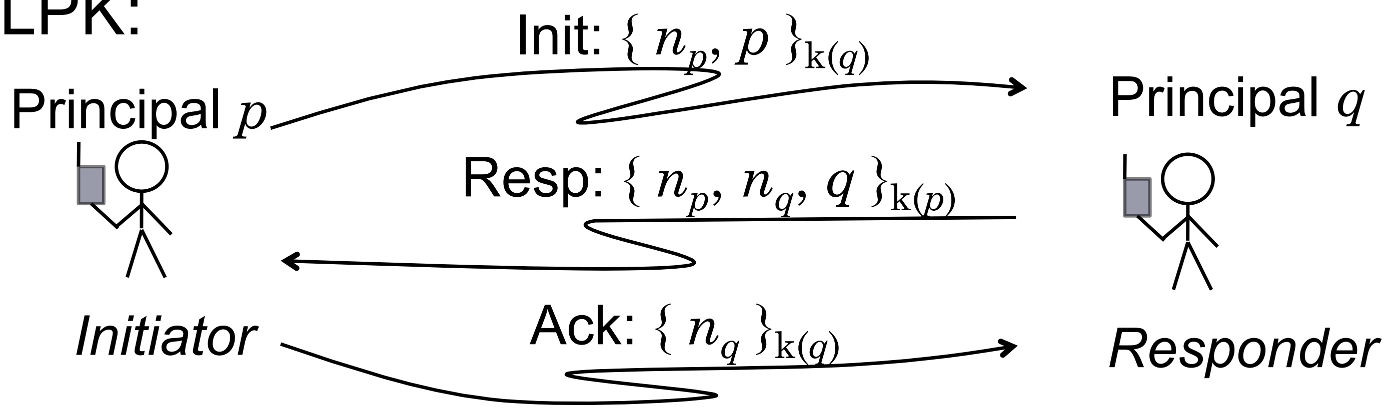
Lecture Note 07
CafeOBJ Team for JAIST-FSSV2010

Topics

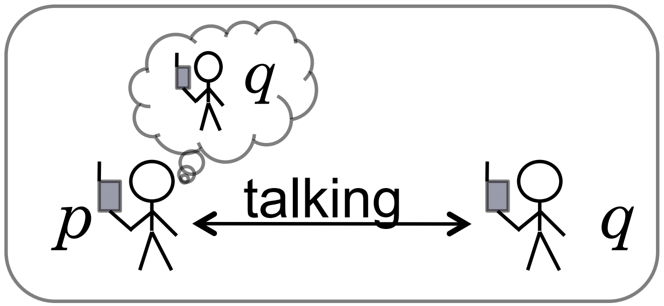
- Brushup of the previous lecture
- Verification that (an abstract model of) NSLPK enjoys Agreement property
- proof score templates
- Case analysis & lemma conjecture

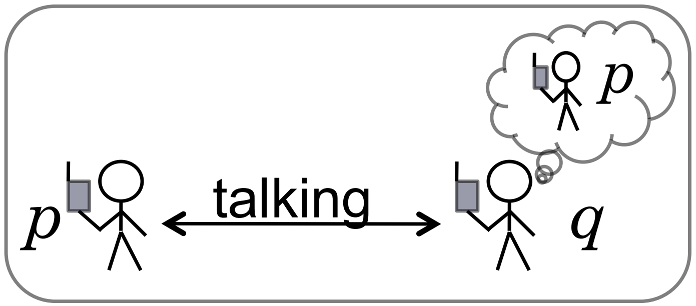
Brushup (1)

- NSLPK:



- Agreement Property: Whenever a protocol run is successfully completed by p and q,
  - the principal with which p is communicating is really q, and
  - the principal with which q is communicating is really p.





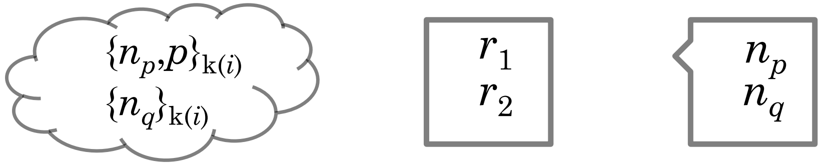
Brushup (2)

- Nonces: $\mathtt{n}(p, q, r)$ denotes a nonce made by p for q, where r makes it unique and unguessable.
- Messages: $\mathtt{m}i(p?, p, q, e_i)$ $(i = 1,2,3)$ denotes a message (an Init, Resp, or Ack message) that seems to have been sent by p to q but has been created by $p?$, which may not be p, where e_i is the message body (ciphertext).
- Networks: Formalized as soups of messages.
  - Sending a message is formalized as putting it in the soup.
  - If the soup contains $\mathtt{m}i(p?, p, q, e_i)$, then q can receive it.
  - Then q believes that it originates in p, although it is not true.
  - Suppose that messages are never deleted from the soup.

Brushup (3)

- Three observable values:

```
op network : System -> Network
op rands   : System -> RandSoup
op nonces  : System -> NonceSoup
```



- Formalization of sending messages:

```
op sdm1 : System Principal Principal Random -> System {constr}
op sdm2 : System Principal Principal Principal
                                Random Nonce -> System {constr}
op sdm3 : System Principal Principal Principal
                                 Nonce Nonce -> System {constr}
```

- Formalization of faking messages:

```
op fkm11 : System Principal Principal Message1    -> System {constr}
op fkm12 : System Principal Principal Nonce       -> System {constr}
op fkm21 : System Principal Principal Message2    -> System {constr}
op fkm22 : System Principal Principal Nonce Nonce -> System {constr}
op fkm31 : System Principal Principal Message3    -> System {constr}
op fkm32 : System Principal Principal Nonce       -> System {constr}
```

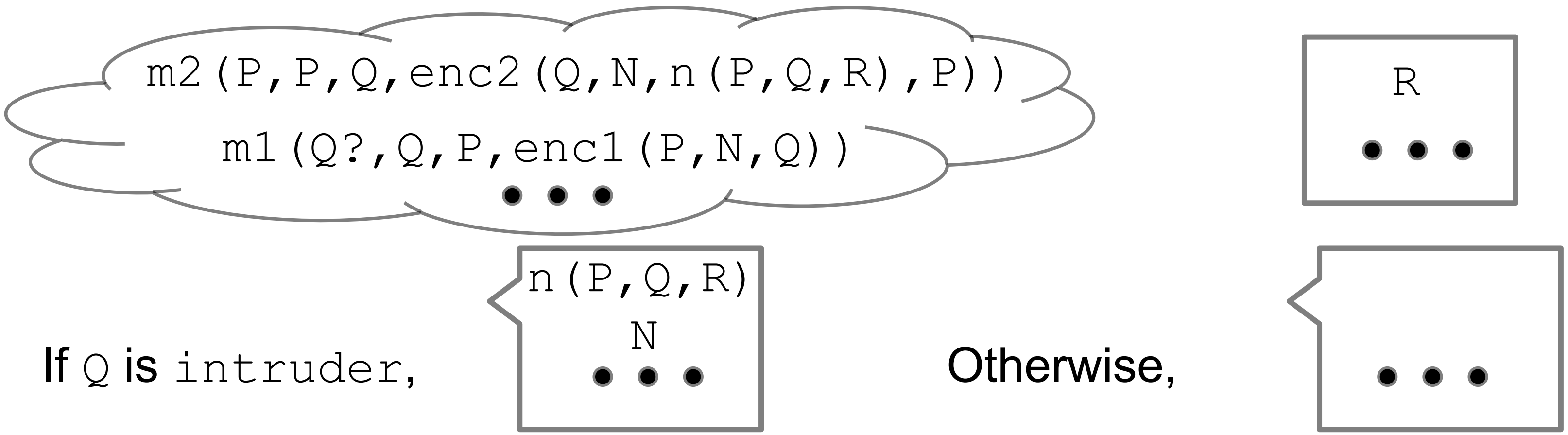
Brushup (4)

- Equations for `sdm2`:

```
ceq network(sdm2(S,Q?,P,Q,R,N))
 = m2(P,P,Q,enc2(Q,N,n(P,Q,R),P)) network(S)if c-sdm2(S,Q?,P,Q,R,N) .
ceq rands(sdm2(S,Q?,P,Q,R,N)) = R rands(S) if c-sdm1(S,P,Q,R) .
ceq nonces(sdm2(S,Q?,P,Q,R,N))
 = (if Q = intruder then N n(P,Q,R) nonces(S) else nonces(S) fi)
 if c-sdm2(S,Q?,P,Q,R,N) .
ceq sdm2(S,Q?,P,Q,R,N) = S if not c-sdm2(S,Q?,P,Q,R,N) .
```

where

```
eq c-sdm2(S,Q?,P,Q,R,N)
 = (m1(Q?,Q,P,enc1(P,N,Q)) \in network(S) and not(R \in rands(S))) .
```



Brushup (5)

- Equations for `fkm22`:

```
ceq network(fkm22(S,P,Q,N1,N2))
 = m2(intruder,P,Q,enc2(Q,N1,N2,P)) network(S)
 if c-fkm22(S,P,Q,N1,N2) .
 eq rands(fkm22(S,P,Q,N1,N2)) = rands(S) .
 eq nonces(fkm22(S,P,Q,N1,N2)) = nonces(S) .
ceq fkm22(S,P,Q,N1,N2)
 = S if not c-fkm22(S,P,Q,N1,N2) .
```

where

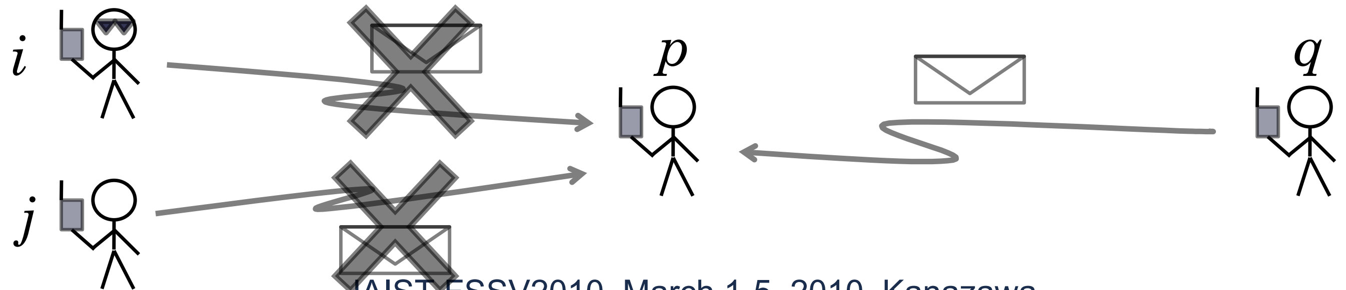
```
eq c-fkm22(S,P,Q,N1,N2)
 = N1 \in nonces(S) and N2 \in nonces(S) .
```

```
m2(intruder,P,Q,enc2(Q,N1,N2,P))
...
```

Brushup (6)

- Formalization of Agreement Property:

```
eq inv1(S,P,Q,Q?,R,N)
 = (not(P = intruder) and
    m1(P,P,Q,enc1(Q,n(P,Q,R),P)) \in network(S) and
    m2(Q?,Q,P,enc2(P,n(P,Q,R),N,Q)) \in network(S)
    implies
    m2(Q,Q,P,enc2(P,n(P,Q,R),N,Q)) \in network(S)) .
eq inv2(S,P,Q,P?,R,N)
 = (not(Q = intruder) and
    m2(Q,Q,P,enc2(P,N,n(Q,P,R),Q)) \in network(S) and
    m3(P?,P,Q,enc3(Q,n(Q,P,R))) \in network(S)
    implies
    m3(P,P,Q,enc3(Q,n(Q,P,R))) \in network(S)) .
```



Preparation for Verification (1)

- Module `PRED-NSLPK`: Properties to verify are declared.

```
mod* PRED-NSLPK {
  inc(NSLPK)
  op inv1 : System Principal Principal Principal
                                  Random Nonce -> Bool
  op inv2 : System Principal Principal Principal
                                  Random Nonce -> Bool
  …
  eq inv1(S,P,Q,Q?,R,N) = … .
  eq inv2(S,P,Q,P?,R,N) = … .
}
```

Preparation for Verification (2)

- Verification starts with use of simultaneous structural induction of `System`.

$$\frac{\left[\;\mathtt{NSLPK} \vdash p_i(\mathtt{init}) \qquad \begin{array}{l}\mathtt{NSLPK} \cup \{p_j(\mathtt{s}) = \mathrm{true}\ \text{ for } j = 1,\ldots,n\} \\ \vdash_{\{\mathtt{s},\mathtt{p},\mathtt{q},\mathtt{r}\}} p_i(\mathtt{sdm1(s,p,q,r)}) \\ \qquad\qquad \bullet\ \bullet\ \bullet\end{array}\;\right] \text{ for } j = 1,\ldots,n}{\mathtt{NSLPK} \vdash (\forall \mathtt{S}:\mathtt{System})\, p_l(\mathtt{S}) \ \text{ for any } l \in \{1,\ldots,n\}}$$

- Module `BASE-NSLPK`: Fresh constants used in proof scores are declared.

```
mod* BASE-NSLPK {  inc(PRED-NSLPK)
  ops s s' : -> System
  op r : -> Random      op n : -> Nonce
  ops p q p? q? : -> Principal
}
```

Preparation for Verification (3)

- Module `ISTEP-NSLPK`: Basic formulas to prove in the induction case (step) and induction hypotheses are declared

An instance of the I.H.

The formula to prove

Induction hypothese

```
mod* ISTEP-NSLPK {  inc(BASE-NSLPK)
  op istep1 : -> Bool
  op istep2 : -> Bool

  eq istep1 =
    inv1(s,p,q,q?,r,n) implies inv1(s',p,q,q?,r,n) .

  eq istep2 =
    inv2(s,p,q,p?,r,n) implies inv2(s',p,q,p?,r,n) .
  "
  eq inv1(s,P,Q,Q?,R,N) = true .
  eq inv2(s,P,Q,P?,R,N) = true .
  "
}
```

Use of Simul Struct Ind of Sort `System`

- The following proof score can be systematically written:

I. Base case:

```
open BASE-NSLPK
-- check
  red inv1(init,p,q,q?,r,n) .
close
```

✓ Done

II. Induction case: For each transition operator t,

```
open ISTEP-NSLPK
-- fresh constants
  op a1 : -> S1 . …
-- assumptions
-- successor state
  eq s′ = t(s,a1,…) .
-- check
  red istep1 .
close
```

Fragments enclosed with open & close in proof scores are called *proof passages*.

```
eq istep1 =
   inv1(s,p,q,q?,r,n)
   implies
   inv1(s',p,q,q?,r,n) .
```

Case Splitting on the Effective Condition

- If t has a non-trivial effective condition c−t, the case is split into two sub-cases based on c−t.

```
open ISTEP-NSLPK
-- fresh constants
  op x1 : -> S1 . …
-- assumptions
  eq c-t(s,x1,…) = true .
-- successor state
  eq s' = t(s,x1,…) .
-- check
  red istep1 .
close
```

```
open ISTEP-NSLPK
-- fresh constants
  op x1 : -> S1 . …
-- assumptions
  eq c-t(s,x1,…) = false .
-- successor state
  eq s' = t(s,x1,…) .
-- check
  red istep1 .
close
```

✓ Done

Transformation of Equations

$$\texttt{eq c-}t\texttt{(S,}X_1\texttt{,...) = } \underbrace{\boxed{C_1\texttt{(S,}X_1\texttt{,...) and ... and } C_n\texttt{(S,}X_1\texttt{,...)}}}_{C\texttt{(S,}X_1\texttt{,...)}}\texttt{.}$$

✓ C`(S,`x_1`,...)` may not be derived from `c-`t`(S,`x_1`,...) = true` with rewriting.

✓ Moreover, each C_i`(S,`x_1`,...)` may not be derived from C`(S,`x_1`,...) = true` with rewriting.

✓ The left proof passage on the previous page is transformed into:

```
open ISTEP-NSLPK
-- fresh constants
  op x1 : -> S1 . ...
-- assumptions
  -- eq c-t(s,x1,...) = true .
  eq C1(s,x1,...) = true .
  ...
  eq Cn(s,x1,...) = true .
  --
-- successor state
  eq s' = t(s,x1,...) .
-- check
  red istep1 .
close
```

Use of "Introduction of `not`"

- Some more transformation (1):
  - ✓ In the induction case for `sdm1` where `c-sdm1(s,…)` holds:

```
eq not(r10 \in rands(s)) = true .
```

derived by rewriting ↑ ⬇ transform not derived by rewriting

```
eq r10 \in rands(s) = false .
```

$$\frac{S \cup \{\mathrm{q} = \texttt{false}\} \vdash p}{S \cup \{\texttt{not}\ \mathrm{q} = \texttt{true}\} \vdash p}$$

Use of “Elimination of Soup Constructor 1”

- Some more transformation (2):
  - ✓ In the induction case for `sdm2` where `c-sdm2(s,…)` holds:

```
eq m1(q10?,q10,p10,enc1(p10,n10,q10))
   \in network(s) = true .
```

derived by rewriting

transform

not derived by rewriting

```
op nw10 : -> Network .
eq network(s)
   = m1(q10?,q10,p10,enc1(p10,n10,q10)) nw10 .
```

$$\frac{\mathrm{S} \cup \{soup = elt\ s\} \vdash_{\{s\}} p}{\mathrm{S} \cup \{elt\ \texttt{\textbackslash in}\ soup = \texttt{true}\} \vdash p}$$ **if** S includes `SOUP`

Use of “Elimination of Soup Constructor 2”

- Some more transformation (3):
  - ✓ In the induction case for `fkm22` where `c-fkm22(s,…)` holds:

```
eq n10 \in nonces(s) = true .
eq n20 \in nonces(s) = true .
eq (n10 = n20) = false .
```

derived by rewriting ↑ | ⇓ transform | not derived by rewriting

```
op ns10 : -> NonceSoup .
eq nonces(s) = n10 n20 ns10 .
eq (n10 = n20) = false .
```

$$\frac{S \cup \{soup = elt_1\ elt_2\ s\ , (elt_1 = elt_2) = \texttt{false}\} \vdash_{\{s\}} p}{S \cup \{elt_1\ \texttt{\textbackslash in}\ soup = \texttt{true},\ elt_2\ \texttt{\textbackslash in}\ soup = \texttt{true}, (elt_1 = elt_2) = \texttt{false}\} \vdash p}$$ **if** S includes `SOUP`

Replacement of Equation with Lemma

$$\frac{\mathrm{S} \cup \{soup = elt\ s\} \mid\!-_{\{s\}}\ p}{\mathrm{S} \cup \{elt \ \texttt{\textbackslash in}\ soup = \texttt{true}\} \mid\!-\ p} \quad \textbf{if } \mathrm{S} \text{ includes } \texttt{SOUP}$$

✓ This is an instance of the following proof rule:

$$\frac{\mathrm{S} \cup \{l_2[X \leftarrow a] = r_2[X \leftarrow a]\} \mid\!-_{\{a\}}\ p}{\mathrm{S} \cup \{l_1 = r_1\} \mid\!-\ p} \quad \textbf{if } \mathrm{S} \mid\!-\ (\exists X)(l_2(X) = r_2(X)) \text{ if } l_1 = r_1$$

Let $l_1 = r_1$ be $elt \ \texttt{\textbackslash in}\ soup = \texttt{true}$ and $l_2(X) = r_2(X)$ be $soup = elt\ s$.

$$\mathrm{S} \mid\!-\ (\exists C)(soup = elt\ C) \text{ if } elt \ \texttt{\textbackslash in}\ soup$$

Preferable Equations

- Assume that two sets $\boldsymbol{E}_1$, $\boldsymbol{E}_2$ of equations are equivalent in a proof passage. If each equation in $\boldsymbol{E}_2$ can be derived from $\boldsymbol{E}_1$ (together with the equations available in the proof passage) with rewriting, then $\boldsymbol{E}_1$ is preferable to $\boldsymbol{E}_2$.

If CafeOBJ does not return `true` *for a proof passage, try to find a set of equations that is preferable to the set of equations used in the proof passage as assumptions and use it.*

Proof Score Templates

- The proof score obtained so far is called a *proof score template*. (See file `template.mod`.)
- The proof score template can be used to verify any (invariant) properties of the specification of NSLPK.
- Proof scores templates can be systematically written for specifications of OTSs.

For verification that an OTS enjoys some properties, to begin with, write a proof score template!

Form of Effective Conditions

- Assumption on the form of effective conditions: Although any forms can be used, the recommended form is a conjunction of literals.

```
eq c-sdm2(S,Q?,P,Q,R,N)
 = (m1(Q?,Q,P,enc1(P,N,Q)) \in network(S) and
   not(R \in rands(S))) .
```

✓ If you want to use a different form such as

$(C_1(\mathrm{S}, X_1, \ldots)$ `or` $C_2(\mathrm{S}, X_1, \ldots))$ `and` $C_3(\mathrm{S}, X_1, \ldots)$,

then convert it into a disjunctive normal form (DNF) such as

$(C_1(\mathrm{S}, X_1, \ldots)$ `and` $C_3(\mathrm{S}, X_1, \ldots))$ `or` $(C_2(\mathrm{S}, X_1, \ldots)$ `and` $C_3(\mathrm{S}, X_1, \ldots))$

and use the same number of transition operators as that of the conjuncts in the DNF such as two.

Induction Case for `fkm21` (1)

- Let us consider the case where `c-fkm21(s,…)` holds.

✓ CafeOBJ does not return any results.

✓ So, let us look at the formula to prove

```
  inv1(s',p,q,q?,r,n)
```

which contains

```
  not(P = intruder)
```

in the premise.

✓ Then, this is used to split the case into two sub-cases.

```
open ISTEP-NSLPK
-- fresh constants
  ops p10 q10 : -> Principal .
  op m20 : -> Message2 .
  op nw10 : -> Network .
-- assumptions
  -- eq c-fkm21(s,…) = true .
  eq network(s) = m20 nw10 .
  --
-- successor state
  eq s' = fkm21(s,p10,q10,m20) .
-- check
  red istep1 .
close
```

Induction Case for `fkm21` (2)

cases	results
`p = intrude`	`true`
`(p = intruder) = false`	neither `true` nor `false`

```
open ISTEP-NSLPK
-- fresh constants
  ops p10 q10 : -> Principal .
  op m20 : -> Message2 .
  op nw10 : -> Network .
-- assumptions
  -- eq c-fkm21(s,…) = true .
  eq network(s) = m20 nw10 .
  --
  eq p = intruder .
-- successor state
  eq s' = fkm21(s,p10,q10,m20) .
-- check
  red istep1 .
close
```

```
open ISTEP-NSLPK
-- fresh constants
  ops p10 q10 : -> Principal .
  op m20 : -> Message2 .
  op nw10 : -> Network .
-- assumptions
  -- eq c-fkm21(s,…) = true .
  eq network(s) = m20 nw10 .
  --
  eq (p = intruder) = false .
-- successor state
  eq s' = fkm21(s,p10,q10,m20) .
-- check
  red istep1 .
close
```

Induction Case for `fkm21` (3)

- The difference of `s` and `s'` that affects the property: `network(s)` and `network(s')`, whose diff. is `m2(intruder,p10,q10,cipher2(m20))`.

- So, the following formula is used to split the latter case (`(p = intruder) = false`) into two sub-cases:

```
m2(intruder,p10,q10,cipher2(m20))
= m2(q?,q,p,enc2(p,n(p,q,r),n,q))
```

A_3

cases	results
A_3	neither `true` nor `false`
A_3 `= false`	`true`

Induction Case for `fkm21` (4)

- In the former case (A_3), instead of one equation, the following four equations are used:

```
eq q? = intruder .
eq p10 = q .
eq q10 = p .
eq cipher2(m20) = enc2(p,n(p,q,r),n,q) .
```

- CafeOBJ returns neither `true` nor `false` for the case.
- We notice that if "`q = intruder`", then "`m2(q?,…)`" in the premise of `inv1(s',p,q,q?,r,n)` equals "`m2(q,…)`" in the conclusion.
- So, "`q = intruder`" is used to split the case into two sub-cases.

Induction Case for `fkm21` (5)

cases	results
`q = intrude`	`true`
`(q = intruder) = false`	neither `true` nor `false`

- For the latter case (`(q = intruder) = false`), we notice that if the formula

  `m1(p,p,q,enc1(q,n(p,q,r),p)) \in nw10` A_5

  does not hold, the premise of `inv1(s',p,q,q?,r,n)` does not hold.

cases	results
A_5	neither `true` nor `false`
A_5 `= false`	`true`

Induction Case for `fkm21` (6)

- For the former case (A_5), we also notice that if the formula

$$\boxed{\texttt{m2(q,q,p,enc2(p,n(p,q,r),n,q)) \textbackslash in nw10}} \quad A_6$$

holds, the conclusion of `inv1(s',p,q,q?,r,n)` holds.

cases	results
A_6	`true`
A_6 `= false`	neither `true` nor `false`

Induction Case for `fkm21` (7)

- The remaining case is characterized by the following equations:

```
network(s) = m20 nw10,
(p = intruder) = false ,
q? = intruder ,  p10 = q ,  q10 = p ,
cipher2(m20) = enc2(p,n(p,q,r),n,q) ,
(q = intruder) = false ,
m1(p,p,q,enc1(q,n(p,q,r),p)) \in nw10 = true,
m2(q,q,p,enc2(p,n(p,q,r),n,q)) \in nw10 = false
```

- We can do further case splitting, but our understanding of NSLPK tells us that there seems to be some contradiction in the set of equations.

Induction Case for `fkm21` (8)

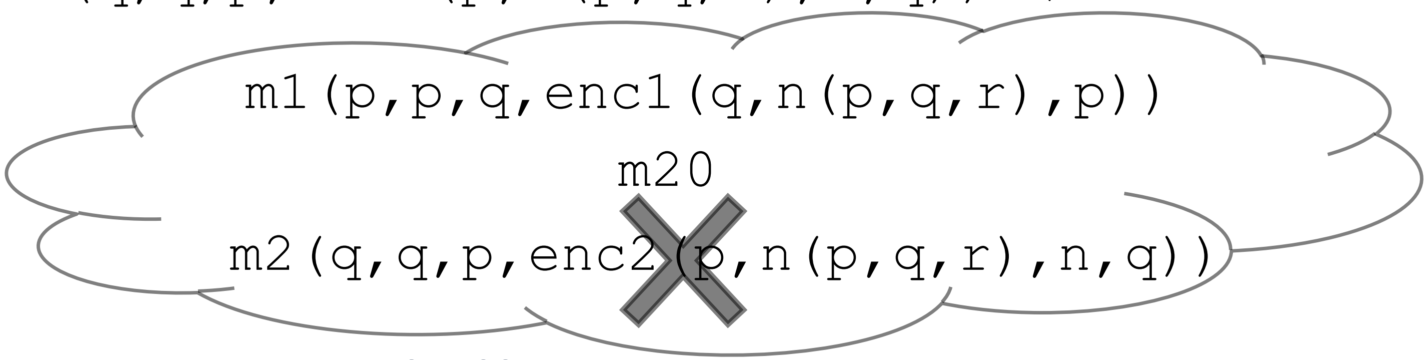
- The assumptions say that
  - There exists a valid Init message really sent by a non-intruder `p` to a non-intruder `q`.

    `m1(p,p,q,enc1(q,n(p,q,r),p)) \in nw10 = true`
  - There exists a Resp message `m20` whose body (ciphertext) is valid as the reply to the Init message.

    `network(s) = m20 nw10`

    `cipher2(m20) = enc2(p,n(p,q,r),n,q)`
  - But, `q` has not replied to the Init message.

    `m2(q,q,p,enc2(p,n(p,q,r),n,q)) \in nw10 = false`



Induction Case for `fkm21` (9)

- These must imply that `m20` has been faked by the intruder.

- To this end, the intruder needs to get either `enc2(p,n(p,q,r),n,q)` or `n(p,q,r)`.
  - It seems impossible to get the former because `q` has not replied to the Init message.
  - It also seems impossible to get the latter because `n(p,q,r)` only appears in `enc1(q,n(p,q,r),p)`, which cannot be decrypted by the intruder.

Induction Case for `fkm21` (10)

- So, if there exist a valid Init message really sent by a non-intruder `p` to a non-intruder `q` and a Resp message `m20` whose body (ciphertext) is valid as the reply to the Init message, then `q` must have replied to the Init message.

```
m1(p,p,q,enc1(q,n(p,q,r),p))
m20
```

implies

```
m1(p,p,q,enc1(q,n(p,q,r),p))
m20
m2(q,q,p,enc2(p,n(p,q,r),n,q))
```

Induction Case for `fkm21` (11)

- A lemma candidate:

`inv4(S,P,Q,N,R,M2)`

```
not(P = intruder) and not(Q = intruder) and
m1(P,P,Q,enc1(Q,n(P,Q,R),P)) \in network(S) and
M2 \in network(S) and
cipher2(M2) = enc2(P,n(P,Q,R),N,Q)
implies
m2(Q,Q,P,enc2(P,n(P,Q,R),N,Q)) \in network(S)
```

- `inv4(S,P,Q,N,R,M2)` can be used to discharge the remaining case:

```
inv4(s,p,q,n,r,m20) implies istep1
```

Lemma Conjecture

- ♦ If you notice a contradiction in the set of equations used in a proof passage, then you can conjecture a lemma.

- ♦ How to notice a contradiction
  - If CafeOBJ returns false, it is most likely that there exists a contradiction.
  - If you understand your target system reasonably well, you can notice a contradiction.

Try to understand your target system as much as possible!

✓ A verification attempt lets you understand your target system better partly because you need to understand it better.

Lemmas for Verification of `inv1`

- We need two more lemmas:

```
eq inv3(S,M2)
   = (M2 \in network(S)
      implies
      random(nonce1(cipher2(M2))) \in rands(S) and
      random(nonce2(cipher2(M2))) \in rands(S)) .
eq inv5(S,N)
   = (N \in nonces(S)
      implies creator(N) = intruder or
              forwhom(N) = intruder) .
```

  - The latter is what is called (*Nonce*) *Secrecy Property*.

Verification of `inv3`

- We need two lemmas:

```
eq inv8(S,M1)
   = (M1 \in network(S)
      implies
      random(nonce(cipher1(M1))) \in rands(S)) .
eq inv9(S,N)
   = (N \in nonces(S)
      implies random(N) \in rands(S)) .
```

Verification of Secrecy Property (`inv5`)

- We need two lemmas:

```
eq inv11(S,P,N,M1)
  = (M1 \in network(S)
         and cipher1(M1) = enc1(P,N,intruder)
     implies
     creator(N) = intruder
         or forwhom(N) = intruder) .
eq inv12(S,P,N1,N2,M2)
  = (M2 \in network(S)
         and cipher2(M2) = enc2(P,N1,N2,intruder)
     implies
     creator(N2) = intruder
         or forwhom(N2) = intruder) .
```

Verification of `inv2`

- `inv1` is Agreement Property from the initiators' (p's) point of view, while `inv2` from the responders' (q's) point of view.
- Although `inv2` is not exactly symmetric to `inv1` w.r.t. NSLPK, they have some similarities.
- Hence, `inv2` can be proved in a similar way to prove `inv1`.
- The proof of `inv2` uses three lemmas, one of which is Secrecy Property (`inv5`).
- To complete the verification, we need one more lemma.

Other Case Studies on Protocol Verification

- *i*KP (Internet Key Protocol) Electronic Payment Protocol

  K. Ogata, K. Futatsugi: Formal analysis of the *i*KP electronic payment protocols, 1st ISSS, LNCS 2609, Springer, pp.441-460 (2003).
- Horn-Preneel Micropayment Protocol

  K. Ogata, K. Futatsugi: Formal verification of the Horn-Preneel micropayment protocol, 4th VMCAI, LNCS 2575, Springer, pp.238-252 (2003).
- SET (Secure Electronic Transactions) Electronic Payment Protocol

  K. Ogata, K. Futatsugi: Equational Approach to Formal Verification of SET, 4th QSIC, IEEE CS Press, pp.50-59 (2004).
- NetBill Electronic Commerce Protocol

  K. Ogata, K. Futatsugi: Formal Analysis of the NetBill Electronic Commerce Protocol, 2nd ISSS, LNCS 3233, Springer, pp.45-64 (2004).
- TLS (Transaction Layer Security) Authentication Protocol

  K. Ogata, K. Futatsugi: Equational Approach to Formal Analysis of TLS, 25th ICDCS, IEEE CS Press, pp.795-804 (2005).
- Mondex Electronic Purse Protocol

  W. Kong, K. Ogata, K. Futatsugi: Algebraic Approaches to Formal Analysis of the Mondex Electronic Purse System, 6th IFM, LNCS 4591, Springer, pp.393-412 (2007).

Summary

- Verification that NSLPK enjoys Agreement Property has been used as an example to discuss what to do for writing proof scores:
  - First write a proof score template, which can be used for any (invariant) properties.
  - Do case splitting and/or conjecture lemmas to complete a proof score of a property.
  - Use preferable equations in proof passages.
  - Try to understand your target system as much as possible to conjecture (good) lemmas.