Modeling and Specification of Authentication Protocol (NSLPK) in OTS/CafeOBJ

Lecture Note 06

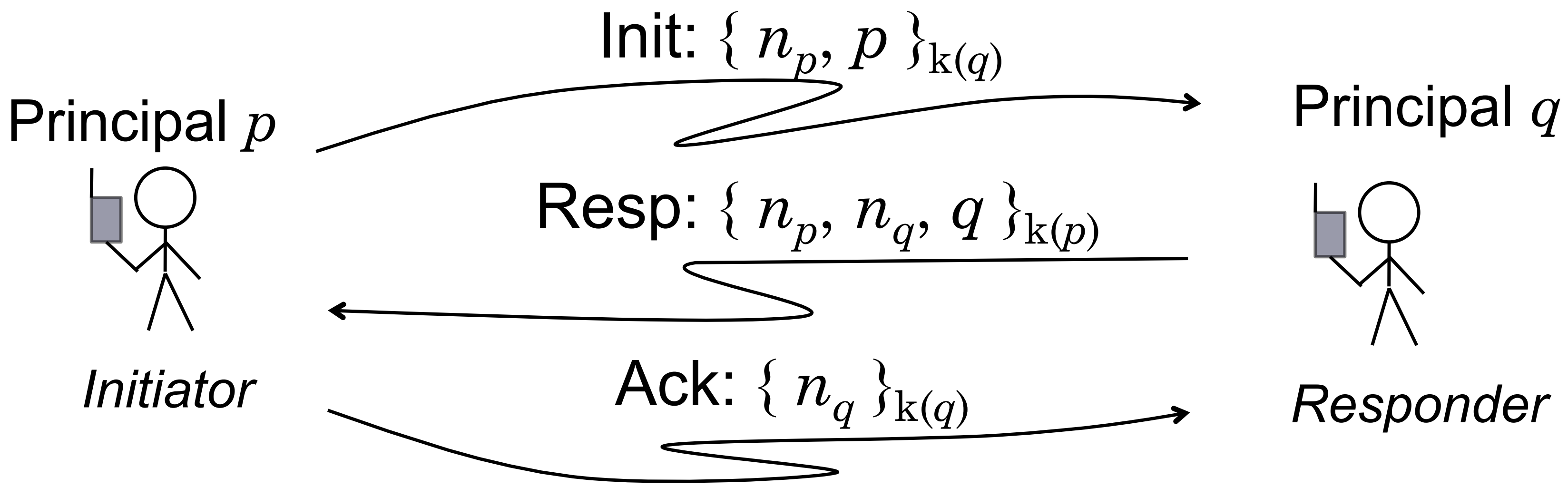
CafeOBJ Team for JAIST-FSSV2010

Topics

- NSLPK, an authentication protocol, and Agreement Property that NSLPK should enjoy.
- A specification of NSLPK.
- Formalization of Agreement Property.

NSLPK

- An authentication protocol that is a revised version (by Gavin Lowe in 1995) of NSPK published by Roger Needham and Michael Schroeder in 1978.

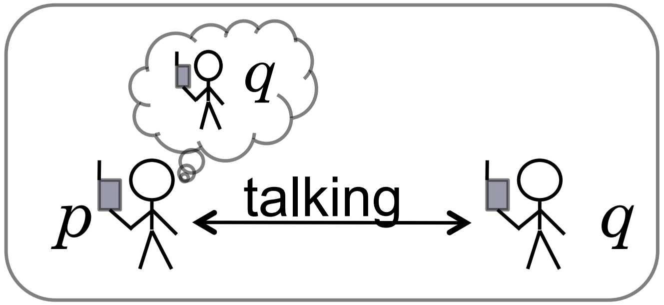


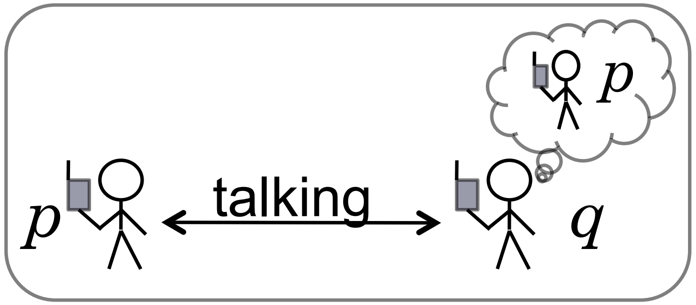
✓ n_x : a nonce (an unguessable random number) made by a pricipal x.
✓ $\{m\}_{\mathrm{k}(x)}$: a ciphertext obtained by encrypting a message (tuple) m with the principal x's public key.

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

  even if there are malicious principals.

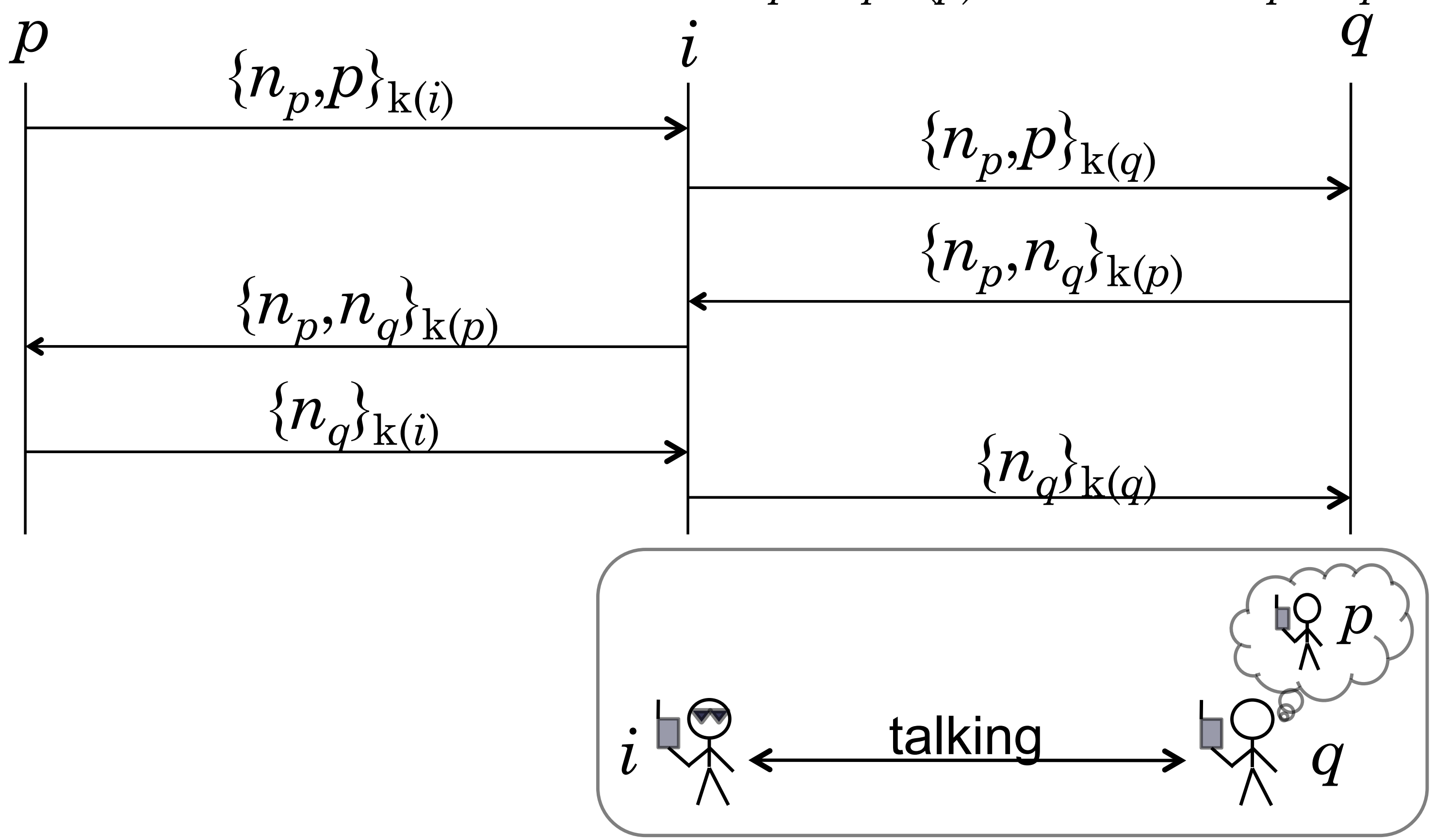




- For verification of the property,
  - the assumptions used are made clear,
  - a transition system (an OTS) of NSLPK is made, and
  - the property is formalized.

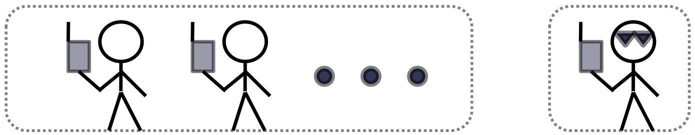
Lowe’s Attack on NSPK

- 17 years passed since it was created till an attack was found by Lowe on NSPK whose difference from NSLPK is only that a Resp message is $\{n_p, n_q\}_{\mathrm{k}(p)}$ but not $\{n_p, n_q, q\}_{\mathrm{k}(p)}$.

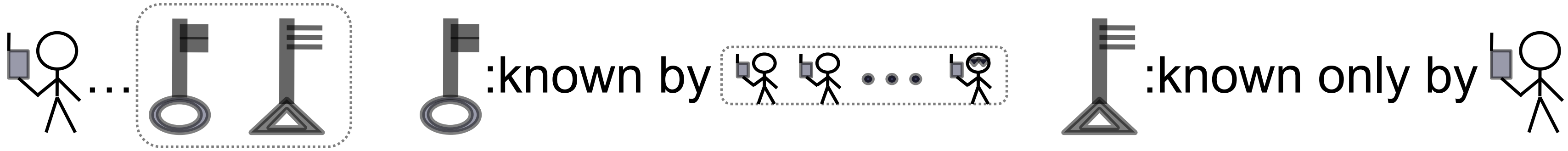


Assumptions (1)

- There are an arbitrary number of principals, all of which except for one are trustable.

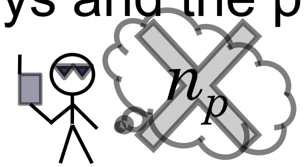
trustable  untrustable

- Each principal is given a pair of public & private keys; the public key is known by all principals, while the private key only by the principal.



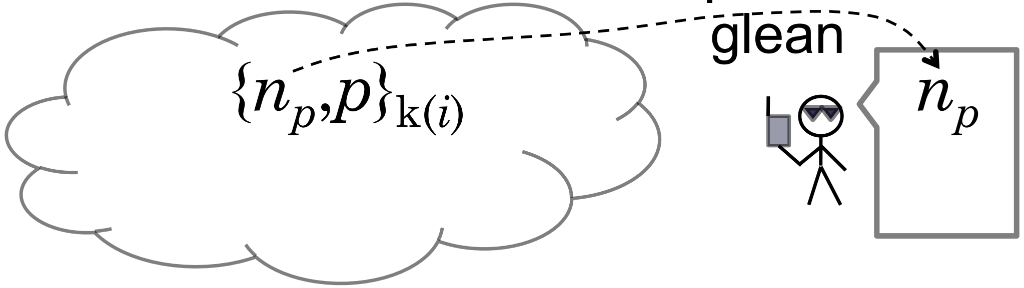
- The cryptosystem used is perfect:
  - Ciphertexts can only be decrypted with the corresponding private keys.

  - Nonces (and private keys and the plaintexts of ciphertexts) cannot be guessed.

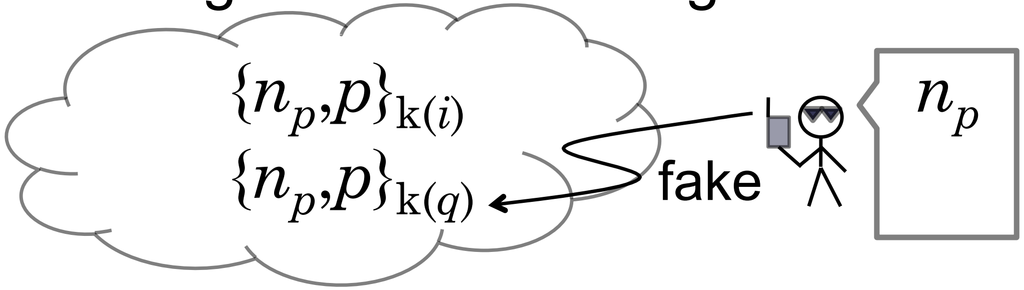


Assumptions (2)

- The behaviors of malicious principals are formalized as the *intruder* proposed by Dolev&Yao; the intruder does the following:
  - to glean as much information as possible from the network,



  - to fake messages based on the gleaned information, and



  - to imitate a trustable principal.

Making an OTS of NSLPK

- Formalization of data such as nonces, ciphertexts, etc.
- Formalization of the behaviors of NSLPK.
  - To determine what values are observed.
  - Formalization of sending messages exactly obeying the protocol.
  - Formalization of faking messages based on the gleaned information from the network.

Principals & Random Numbers

- Module `PRINCIPAL`:

```
mod* PRINCIPAL principal-sort Principal {
  [Principal]
  op intruder : -> Principal
  op _=_ : Principal Principal -> Bool {comm}
  eq (P:Principal = P) = true .
  ceq P1:Principal = P2:Principal if P1 = P2 .
}
```

- Module `RANDOM`:

```
mod* RANDOM principal-sort Random {
  [Random]
  op _=_ : Random Random -> Bool {comm}
  eq (R:Random = R) = true .
  ceq R1:Random = R2:Random if R1 = R2 .
}
```

Nonces

- Module `NONCE`: One constructor is declared.

```
op n : Principal Principal Random -> Nonce {constr}
```

✓ $\texttt{n}(p,q,r)$ denotes a nonce made by p for q, where r makes the nonce unique and unguessable.

- n_p in $\{ n_p, p \}_{\mathrm{k}(q)}$ is denoted by $\texttt{n}(p,q,r_1)$.
- n_q in $\{ n_p, n_q, q \}_{\mathrm{k}(p)}$ is denoted by $\texttt{n}(q,p,r_2)$.

✓ p, q, r in $\texttt{n}(p,q,r)$ are meta-information.

✓ The following operators are prepared:

Init:	$p \rightarrow q$	$\{n_p,p\}_{\mathrm{k}(q)}$
Resp:	$q \rightarrow p$	$\{n_p,n_q,q\}_{\mathrm{k}(p)}$
Ack:	$p \rightarrow q$	$\{n_q\}_{\mathrm{k}(q)}$

```
eq creator(n(C,W,R)) = C .
eq forwhom(n(C,W,R)) = W .
eq random(n(C,W,R)) = R .
eq (N1 = N2) = (creator(N1) = creator(N2) and
                forwhom(N1) = forwhom(N2) and
                random(N1) = random(N2)) .
```

Ciphertexts in Init Messages

- Module `CIPHER1`: One constructor is declared.

```
op enc1 : Principal Nonce Principal
                                    -> Cipher1 {constr}
```

  - ✓ `enc1(`p`,`n`,`q`)` denotes a ciphertext $\{ n, q \}_{k(p)}$.
  - ✓ The following operators are provided:

Init:	$p \rightarrow q$	$\{n_p, p\}_{k(q)}$
Resp:	$q \rightarrow p$	$\{n_p, n_q, q\}_{k(p)}$
Ack:	$p \rightarrow q$	$\{n_q\}_{k(q)}$

```
eq key(enc1(K,N,P)) = K .
eq nonce(enc1(K,N,P)) = N .
eq principal(enc1(K,N,P)) = P .
eq (E11 = E12) = (key(E11) = key(E12) and
                  nonce(E11) = nonce(E12) and
                  principal(E11) = principal(E12)) .
```

Ciphertexts in Resp Messages

- Module `CIPHER2`: One constructor is declared.

```
op enc2 : Principal Nonce Nonce Principal
                                  -> Cipher2 {constr}
```

  - ✓ `enc2(`p`,` n_1`,`n_2`,`q`)` denotes a ciphertext $\{ n_1, n_2, q \}_{k(p)}$.
  - ✓ The following operators are provided:

Init:	$p \to q$	$\{n_p,p\}_{k(q)}$
Resp:	$q \to p$	$\{n_p,n_q,q\}_{k(p)}$
Ack:	$p \to q$	$\{n_q\}_{k(q)}$

```
eq key(enc2(K,N1,N2,P)) = K .
eq nonce1(enc2(K,N1,N2,P)) = N1 .
eq nonce2(enc2(K,N1,N2,P)) = N2 .
eq principal(enc2(K,N1,N2,P)) = P .
eq (E21 = E22) = (key(E21) = key(E22) and
                  nonce1(E21) = nonce1(E22) and
                  nonce2(E21) = nonce2(E22) and
                  principal(E21) = principal(E22)) .
```

Ciphertexts in Ack Messages

- Module `CIPHER3`: One constructor is declared.

```
op enc3 : Principal Nonce -> Cipher3 {constr}
```

✓ `enc3(`p`,` n`)` denotes a ciphertext $\{ n \}_{\mathrm{k}(p)}$.

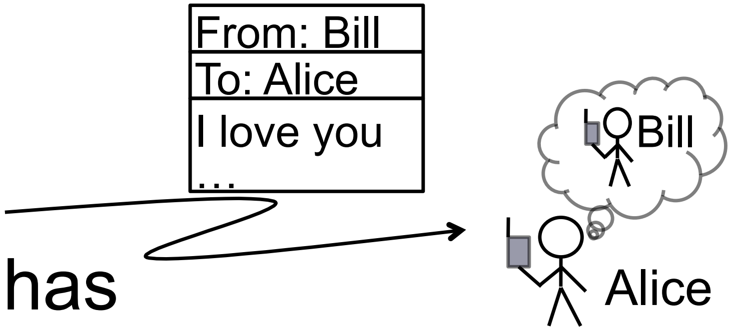
✓ The following operators are provided:

Init:	$p \rightarrow q$	$\{n_p, p\}_{\mathrm{k}(q)}$
Resp:	$q \rightarrow p$	$\{n_p, n_q, q\}_{\mathrm{k}(p)}$
Ack:	$p \rightarrow q$	$\{n_q\}_{\mathrm{k}(q)}$

```
eq key(enc3(K,N)) = K .
eq nonce(enc3(K,N)) = N .
eq (E31 = E32) = (key(E31) = key(E32) and
                  nonce(E31) = nonce(E32)) .
```

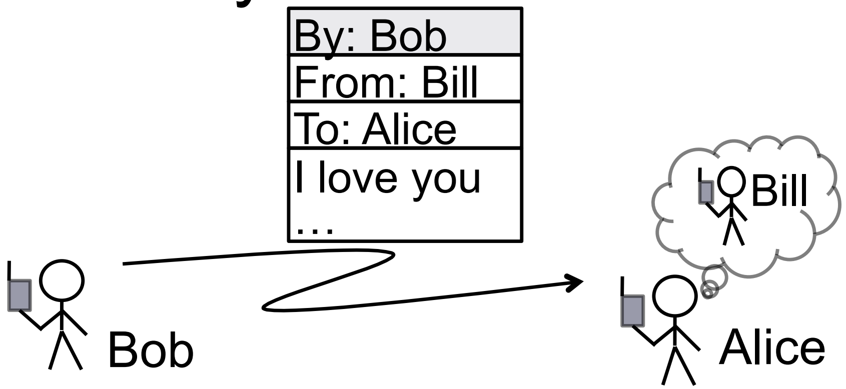
Messages (1)

- A term denoting a message contains
  - the (seeming) source (*sender*),
  - the destination (*receiver*), and
  - the body (*ciphertext*).



- In addition to those data, it also has
  - the actual source (*creator*).

because messages may be faked by the intruder.



This is meta-information.

- Such a term is in the form
  - $m(creator, sender, receiver, ciphertext)$

Messages (2)

- Module `MESSAGE`: Three constructors are declared.

```
[Message1 Message2 Message3 < Message]
op m1 : Principal Principal Principal Cipher1
                                     -> Message1 {constr}
op m2 : Principal Principal Principal Cipher2
                                     -> Message2 {constr}
op m3 : Principal Principal Principal Cipher3
                                     -> Message3 {constr}
```

✓ `m1(`$p?$`,` p`,` q`, enc1(…))` denotes an Init message.
✓ `m2(`$p?$`,` p`,` q`, enc2(…))` denotes a Resp message.
✓ `m3(`$p?$`,` p`,` q`, enc3(…))` denotes an Ack message.

Init:	$p \rightarrow q$	$\{n_p,p\}_{\mathrm{k}(q)}$
Resp:	$q \rightarrow p$	$\{n_p,n_q,q\}_{\mathrm{k}(p)}$
Ack:	$p \rightarrow q$	$\{n_q\}_{\mathrm{k}(q)}$

Messages (3)

- ✓ The following operators are provided:

```
eq creator(m1(C,S,R,E1)) = C .
eq creator(m2(C,S,R,E2)) = C .
eq creator(m3(C,S,R,E3)) = C .
eq sender(m1(C,S,R,E1)) = S .
eq sender(m2(C,S,R,E2)) = S .
eq sender(m3(C,S,R,E3)) = S .
eq receiver(m1(C,S,R,E1)) = R .
eq receiver(m2(C,S,R,E2)) = R .
eq receiver(m3(C,S,R,E3)) = R .
eq cipher1(m1(C,S,R,E1)) = E1 .
eq cipher2(m2(C,S,R,E2)) = E2 .
eq cipher3(m3(C,S,R,E3)) = E3 .
```

- ✓ Note that `cipher`*i* (*i* = `1`,`2`,`3`) is declared as follows:

```
op cipheri : Messagei -> Cipheri
```

Messages (4)

- ✓ The following operators are provided (cont.):

```
eq (M = M) = true .
eq (M11 = M12) = (creator(M11) = creator(M12) and
                  sender(M11) = sender(M12) and
                  receiver(M11) = receiver(M12) and
                  cipher1(M11) = cipher1(M12)) .
eq (M21 = M22) = (creator(M21) = creator(M22) and
                  sender(M21) = sender(M22) and
                  receiver(M21) = receiver(M22) and
                  cipher2(M21) = cipher2(M22)) .
eq (M31 = M32) = (creator(M31) = creator(M32) and
                  sender(M31) = sender(M32) and
                  receiver(M31) = receiver(M32) and
                  cipher3(M31) = cipher3(M32)) .
eq (M11 = M21) = false .
eq (M11 = M31) = false .
eq (M21 = M31) = false .
```

```
var M : Message
vars M11 M12 : Message1
vars M21 M22 : Message2
vars M31 M32 : Message3
```

Soups (Associative & Commutative Collections)

- Module `SOUP`:

```
mod* SOUP (D :: EQTRIV) principal-sort Soup {
  [Elt.D < Soup]
  op empty : -> Soup {constr}
  op _ _ : Soup Soup -> Soup
                        {constr assoc comm id: empty}
  op _\in_ : Elt.D Soup -> Bool
  var S : Soup  vars E1 E2 : Elt.D
  eq E1 \in empty = false .
  eq E1 \in (E2 S) = (E1 = E2) or E1 \in S .
}
```

where `EQTRIV` is as follows:

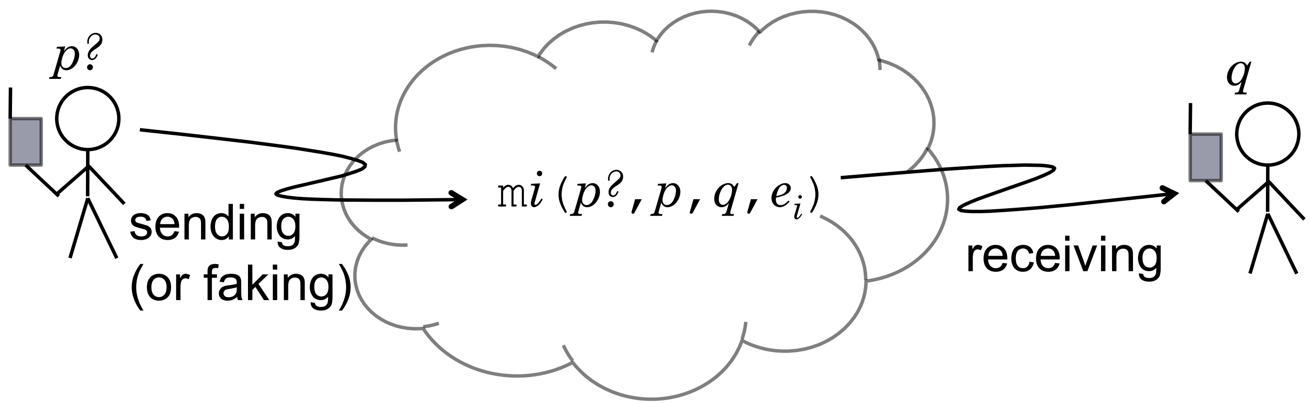
```
mod* EQTRIV principal-sort Elt {
  [Elt]
  op _=_ : Elt Elt -> Bool {comm}
  eq (E:Elt = E) = true .
  eq E1:Elt = E2:Elt if E1 = E2 .
}
```

Networks

- Formalized as soups of messages.

```
SOUP(MESSAGE)*{sort Soup -> Network}
```

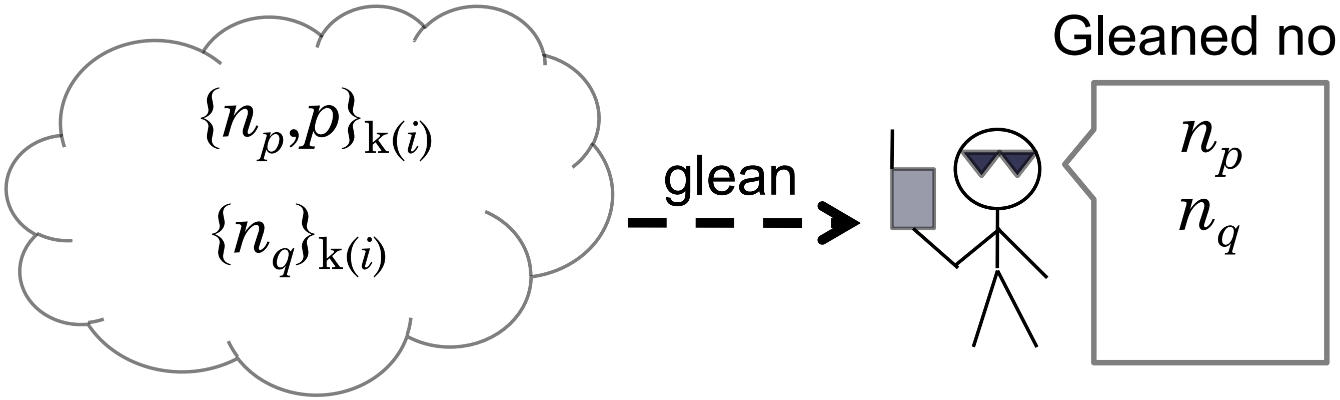
✓ Sending a message is formalized as putting it in the soup.
✓ If the soup contains $\mathtt{mi}(p?, p, q, e_i)$, then q can receive it.
✓ q may believe that it originates in p, but it may not be true.
✓ Suppose that messages are never deleted from the soup.
✓ This assumption may make it possible to do something that can never happen in the real world, but *covers all possible cases*.



Soups of Nonces & Soups of Random Numbers

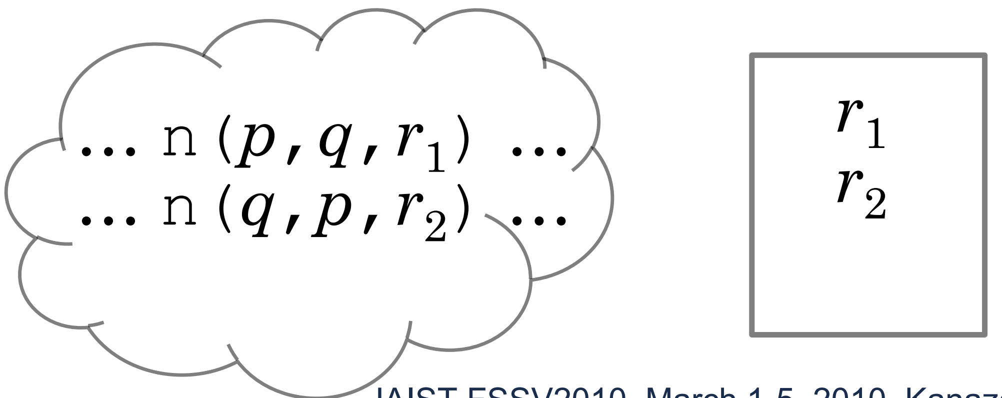
- Soups of nonces:

```
SOUP(NONCE)*{sort Soup -> NonceSoup}
```



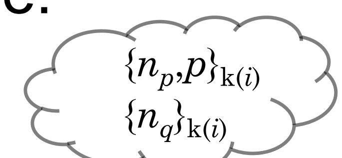
- Soups of random numbers:

```
SOUP(RANDOM)*{sort Soup -> RandSoup}
```

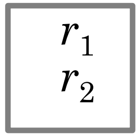


Observable Values

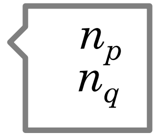
- The three values are made observable:
  - the network (a soup of messages),



  - a soup of random numbers that have been used, and



  - a soup of nonces that have been gleaned by the intruder from the network.



- The corresponding operators (called *observation* or *observer operators* or *functions*) are as follows:

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

where `System` is the sort for the set of states, i.e. the state space.

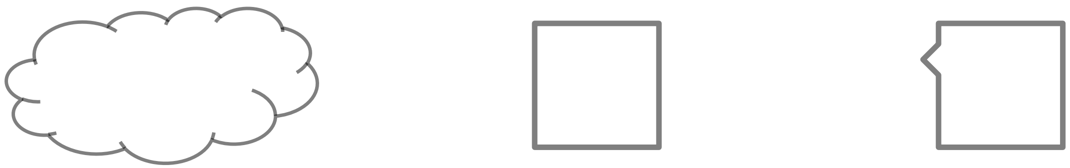
Initial States

- An arbitrary initial state is denoted by the operator:

```
op init : -> System {constr}
```

such that

```
eq network(init) = empty .
eq rands(init) = empty .
eq nonces(init) = empty .
```



Formalization of Sending Messages

- Sending Init, Resp and Ack messages according to the protocol is formalized as the operators (called *transition operators* or *functions*):

```
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}
```

✓ `sdm1`(s,p,q,r) denotes the successor state of s when p sends an Init message to q in s.
✓ `sdm2`$(s,q?,p,q,r,n)$ denotes the successor state of s when p sends a Resp message to q in s.
✓ `sdm3`$(s,q?,p,q,n_1,n_2)$ denotes the successor state of s when p sends an Ack message to p in s.

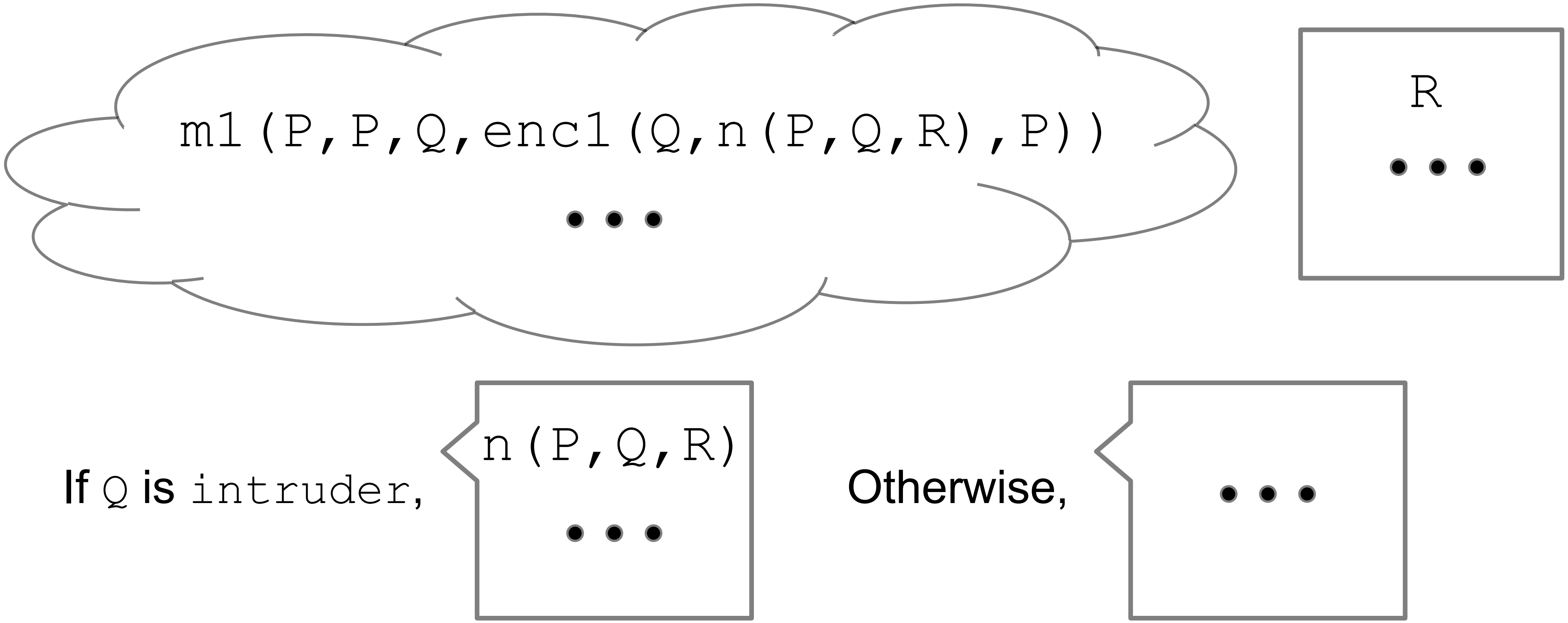
Init:	$p \rightarrow q$	$\{n_p, p\}_{\mathrm{k}(q)}$
Resp:	$q \rightarrow p$	$\{n_p, n_q, q\}_{\mathrm{k}(p)}$
Ack:	$p \rightarrow q$	$\{n_q\}_{\mathrm{k}(q)}$

Sending Init Messages (1)

- The effective condition of `sdm1` is that a random number is fresh.

```
eq c-sdm1(S,P,Q,R) = not(R \in rands(S)) .
```

- When `c-sdm1(S,P,Q,R)` holds, in the successor state `sdm1 (S,P,Q,R)`,



Sending Init Messages (2)

Init:	$p \to q$	$\{n_p,p\}_{k(q)}$
Resp:	$q \to p$	$\{n_p,n_q,q\}_{k(p)}$
Ack:	$p \to q$	$\{n_q\}_{k(q)}$

- The equations for `sdm1`:

```
ceq network(sdm1(S,P,Q,R))
 = m1(P,P,Q,enc1(Q,n(P,Q,R),P)) network(S)
 if c-sdm1(S,P,Q,R) .
ceq rands(sdm1(S,P,Q,R))
 = R rands(S) if c-sdm1(S,P,Q,R) .
ceq nonces(sdm1(S,P,Q,R))
 = (if Q = intruder then n(P,Q,R) nonces(S)
                     else nonces(S) fi)
 if c-sdm1(S,P,Q,R) .
ceq sdm1(S,P,Q,R) = S if not c-sdm1(S,P,Q,R) .
```

Note that we need to have the last one because we have to explicitly declare that nothing changes if `c-sdm1(S,P,Q,R)` does not hold.

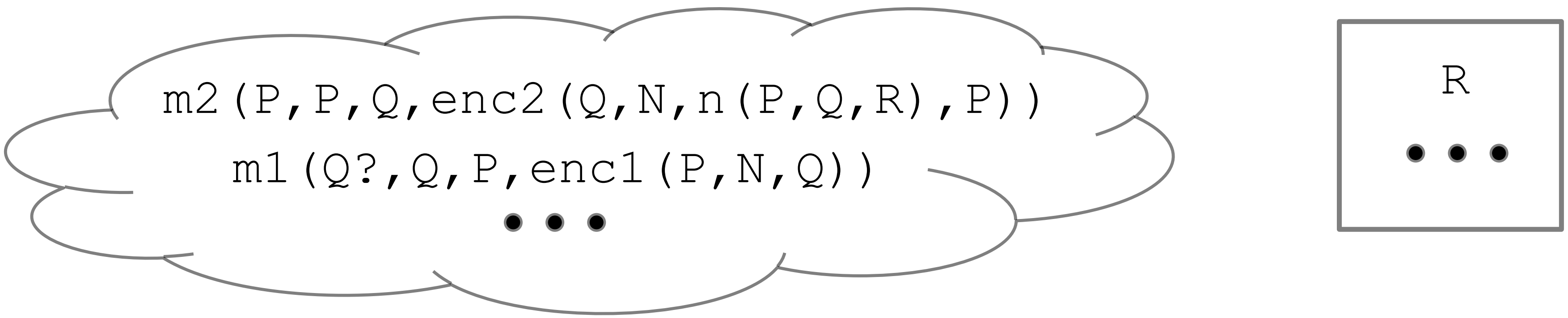
Sending Resp Messages (1)

Init:	$p \rightarrow q$	$\{n_p,p\}_{\mathrm{k}(q)}$
Resp:	$q \rightarrow p$	$\{n_p,n_q,q\}_{\mathrm{k}(p)}$
Ack:	$p \rightarrow q$	$\{n_q\}_{\mathrm{k}(q)}$

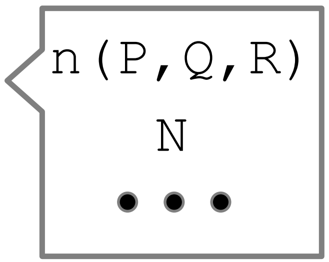
- The effective condition of `sdm2` is that there exists an Init message in the network and a random number is fresh.

```
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))) .
```

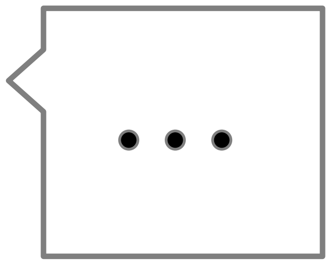
- When `c-sdm2(S,Q?,P,Q,RN)` holds, in the successor state `sdm2(S,Q?,P,Q,R,N)`,



If `Q` is `intruder`,



Otherwise,



Sending Resp Messages (2)

Init:	$p \to q$	$\{n_p, p\}_{\mathrm{k}(q)}$
Resp:	$q \to p$	$\{n_p, n_q, q\}_{\mathrm{k}(p)}$
Ack:	$p \to q$	$\{n_q\}_{\mathrm{k}(q)}$

- The 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) .
```

Sending Ack Messages (1)

Init:	$p \to q$	$\{n_p, p\}_{\mathrm{k}(q)}$
Resp:	$q \to p$	$\{n_p, n_q, q\}_{\mathrm{k}(p)}$
Ack:	$p \to q$	$\{n_q\}_{\mathrm{k}(q)}$

- The effective condition of `sdm3` is that there exist an Init message and a Resp message in the network.

```
eq c-sdm3(S,Q?,P,Q,N1,N2)
 = m1(P,P,Q,enc1(Q,N1,P)) \in network(S) and
   m2(Q?,Q,P,enc2(P,N1,N2,Q)) \in network(S) .
```

- When `c-sdm3(S,Q?,P,Q,N1,N2)` holds, in the successor state `sdm3(S,Q?,P,Q,N1,N2)`,

```
m3(P,P,Q,enc3(Q,N2))
m1(P,P,Q,enc1(Q,N1,P))
m2(Q?,Q,P,enc2(P,N1,N2,Q))
• • •
```

• • •

If `Q` is `intruder`,

N2
• • •

Otherwise,

• • •

Init:	$p \rightarrow q$	$\{n_p, p\}_{k(q)}$
Resp:	$q \rightarrow p$	$\{n_p, n_q, q\}_{k(p)}$
Ack:	$p \rightarrow q$	$\{n_q\}_{k(q)}$

Sending Ack Messages (2)

- The equations for `sdm3`:

```
ceq network(sdm3(S,Q?,P,Q,N1,N2))
 = m3(P,P,Q,enc3(Q,N2)) network(S)
 if c-sdm3(S,Q?,P,Q,N1,N2) .
 eq rands(sdm3(S,Q?,P,Q,N1,N2)) = rands(S) .
ceq nonces(sdm3(S,Q?,P,Q,N1,N2))
 = (if Q = intruder then N2 nonces(S)
                    else nonces(S) fi)
 if c-sdm3(S,Q?,P,Q,N1,N2) .
ceq sdm3(S,Q?,P,Q,N1,N2)
 = S if not c-sdm3(S,Q?,P,Q,N1,N2) .
```

Formalization of Faking Messages

- The intruder may fake messages based on the nonces and ciphertexts gleaned from the network, which is formalized as the transition operators:

```
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}
```

✓ `fkm11(`s`,`p`,`q`,`m_1`)` denotes the successor state of s when the intruder fakes based on m_1 an Init message, which seems to have been sent by p to q, in s.

✓ …

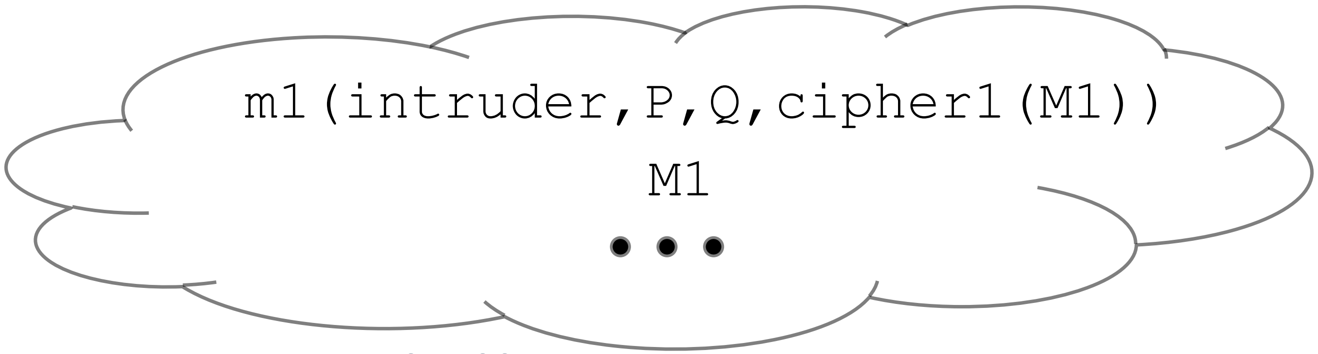
Faking Init Messages (1)

- The effective condition of `fkm11` is that there exists an Init message in the network.

```
eq c-fkm11(S,P,Q,M1) = M1 \in network(S) .
```

- The equations for `fkm11`:

```
ceq network(fkm11(S,P,Q,M1))
 = m1(intruder,P,Q,cipher1(M1)) network(S)
 if c-fkm11(S,P,Q,M1) .
 eq rands(fkm11(S,P,Q,M1)) = rands(S) .
 eq nonces(fkm11(S,P,Q,M1)) = nonces(S) .
ceq fkm11(S,P,Q,M1) = S if not c-fkm11(S,P,Q,M1) .
```



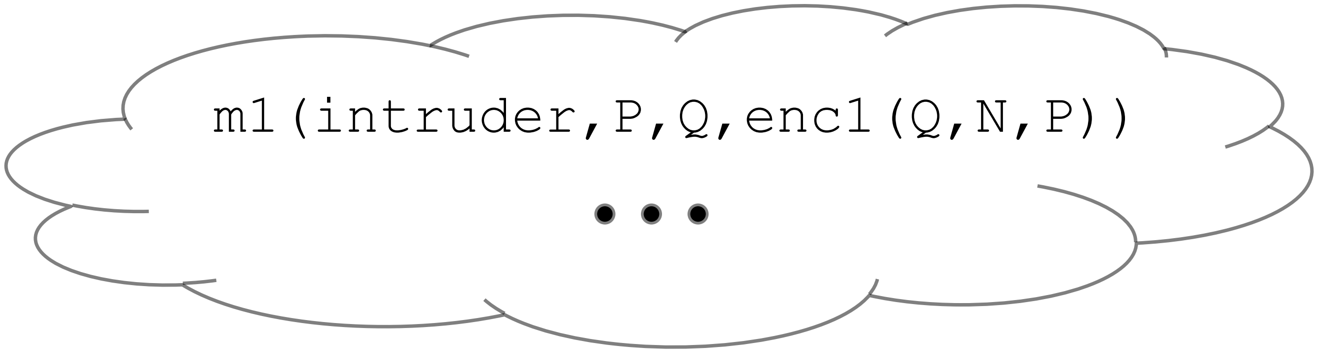
Faking Init Messages (2)

- The effective condition of `fkm12` is that a nonce is available to the intruder.

```
eq c-fkm12(S,P,Q,N) = N \in nonces(S) .
```

- The equations for `fkm12`:

```
ceq network(fkm12(S,P,Q,N))
 = m1(intruder,P,Q,enc1(Q,N,P)) network(S)
 if c-fkm12(S,P,Q,N) .
 eq rands(fkm12(S,P,Q,N)) = rands(S) .
 eq nonces(fkm12(S,P,Q,N)) = nonces(S) .
ceq fkm12(S,P,Q,N) = S if not c-fkm12(S,P,Q,N) .
```



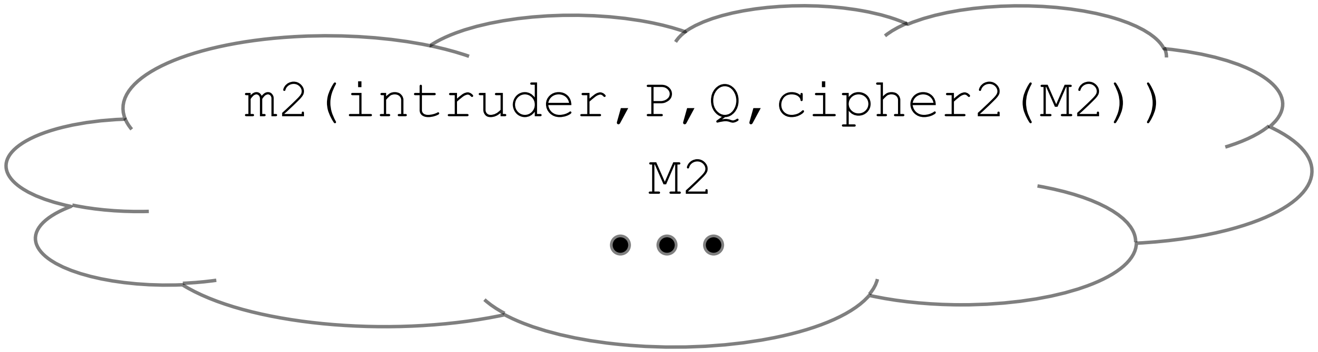
Faking Resp Messages (1)

- The effective condition of `fkm21` is that there exists a Resp message in the network.

```
eq c-fkm21(S,P,Q,M2) = M2 \in network(S) .
```

- The equations for `fkm21`:

```
ceq network(fkm21(S,P,Q,M2))
 = m2(intruder,P,Q,cipher2(M2)) network(S)
 if c-fkm21(S,P,Q,M2) .
 eq rands(fkm21(S,P,Q,M2)) = rands(S) .
 eq nonces(fkm21(S,P,Q,M2)) = nonces(S) .
ceq fkm21(S,P,Q,M2) = S if not c-fkm21(S,P,Q,M2) .
```



Faking Resp Messages (2)

- The effective condition of `fkm22` is that two different nonces are available to the intruder.

```
eq c-fkm22(S,P,Q,N1,N2) = N1 \in nonces(S) and
                  N2 \in nonces(S) and not(N1 = N2) .
```

- The 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) .
```

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

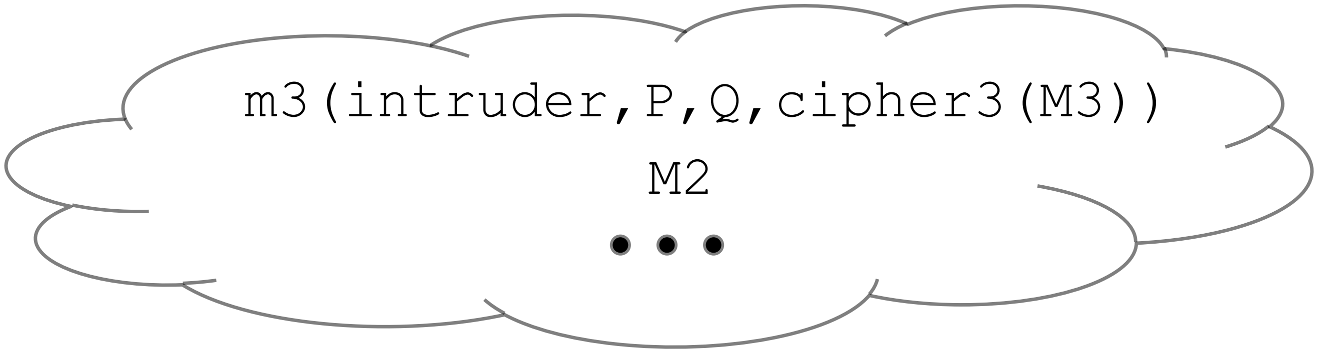
Faking Ack Messages (1)

- The effective condition of `fkm31` is that there exists an Ack message in the network.

```
eq c-fkm31(S,P,Q,M3) = M3 \in network(S) .
```

- The equations for `fkm31`:

```
ceq network(fkm31(S,P,Q,M3))
 = m3(intruder,P,Q,cipher3(M3)) network(S)
 if c-fkm31(S,P,Q,M3) .
 eq rands(fkm31(S,P,Q,M3)) = rands(S) .
 eq nonces(fkm31(S,P,Q,M3)) = nonces(S) .
ceq fkm31(S,P,Q,M3) = S if not c-fkm31(S,P,Q,M3) .
```



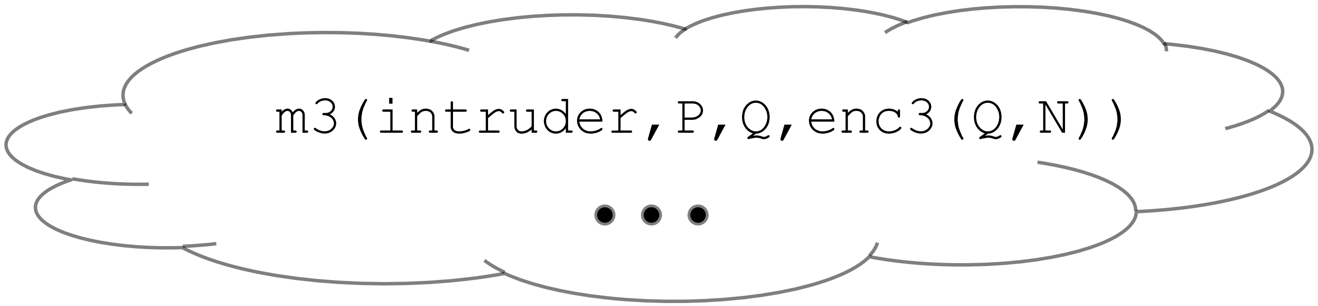
Faking Ack Messages (2)

- The effective condition of `fkm32` is that a nonce is available to the intruder.

```
eq c-fkm32(S,P,Q,N) = N \in nonces(S) .
```

- The equations for `fkm32`:

```
ceq network(fkm32(S,P,Q,N))
 = m3(intruder,P,Q,enc3(Q,N)) network(S)
 if c-fkm32(S,P,Q,N) .
 eq rands(fkm32(S,P,Q,N)) = rands(S) .
 eq nonces(fkm32(S,P,Q,N)) = nonces(S) .
ceq fkm32(S,P,Q,N) = S if not c-fkm32(S,P,Q,N) .
```



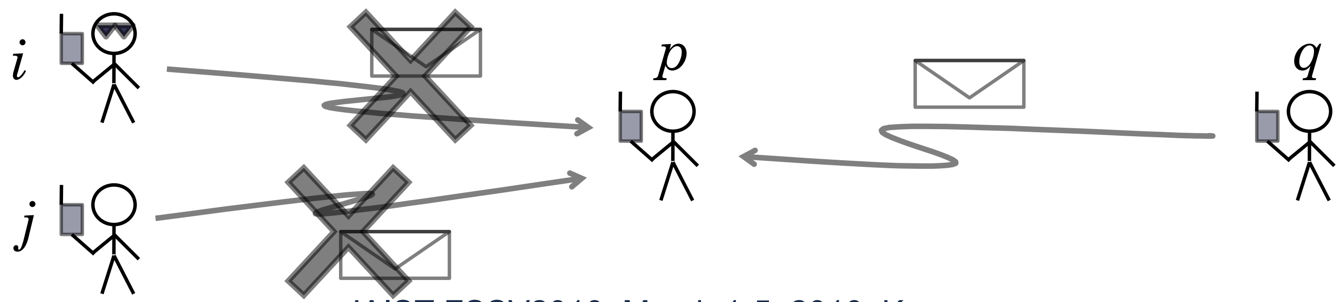
Constructors of State Space

- The (reachable) state space (`System`) is constructed from the constructors `init`, `sdm1`, `sdm2`, `sdm3`, `fkm11`, `fkm12`, `fkm21`, `fkm22`, `fkm31` and `fkm32`.
- Theorems on `System` (or invariant properties of the OTS of NSLPK) can be proved by simultaneous structural induction of the reachable state space (`System`).

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

Formalization of Agreement Property (1)

- 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.
- The property can be rephrased based on the specification of NSLPK as follows:
  - Whenever p receives a Resp message that is what p really expects, the Resp message originates in q, and
  - whenever q receives an Ack message that is what q really expects, the Ack message originates in p.



Formalization of Agreement Property (2)

- Precisely the property is described as follows:
  - If p is not the intruder, then whenever p has sent an Init message to q and receives a valid Resp message, which seems to have been sent by q, the Resp message originates in q, and
  - If q is not the intruder, then whenever q has sent a Resp message to p and receives a valid Ack message, which seems to have been sent by p, the Ack message originates in p.

Note that if p (or q) is the intruder, then the intruder can fake a valid Resp (or Ack) message, which does not originate in q (or p).
If two nonces n,n' are available to the intruder, then the intruder can fake the messages:

```
m1(intruder,intruder,q,enc1(q,n,intruder))
m2(intruder,q,intruder,enc2(intruder,n,n',q))
```

Formalization of Agreement Property (3)

- That there exists a message in the network means that it has been sent by (or originates in) the creator, and any messages in the network whose receivers are p can be received by p. So, the property is formalized:

```
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)) .
```

Summary

- NSLPK has been used as an example to discuss what to prepare for verification that a system enjoys a property:
  - Make the assumptions clear.
  - Create a transition system (an OTS) of the system.
    - ✓ Formalize data used.
    - ✓ Determine what values are observed.
    - ✓ Determine what actions of the system are formalized as transitions.

    You may want to take into account the property.
  - Formalize the property based on the specification of the OTS.