2019

Modelling the Needham Schroeder Protocol

JACOB WILLIAMS

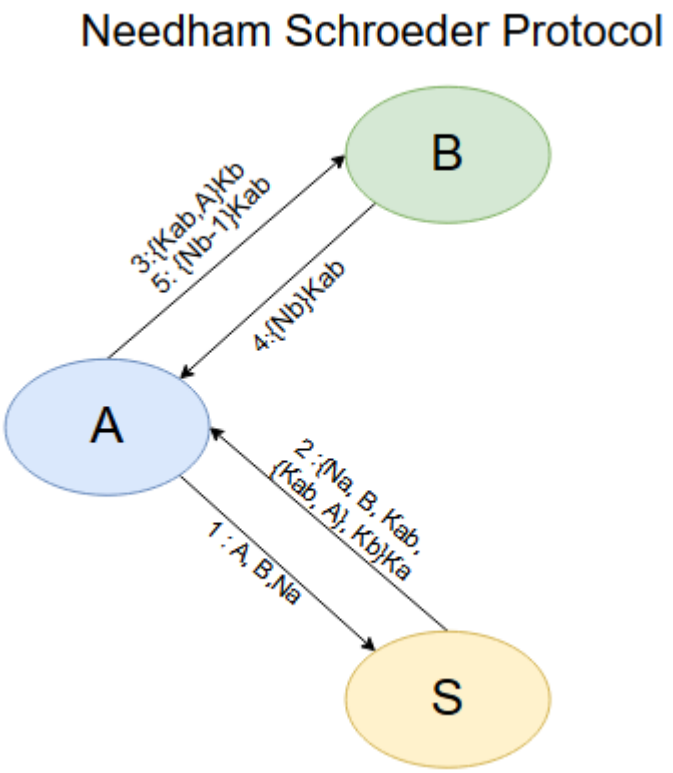
Table of Contents

Diagrams

This section contains all the diagrams utilized in the modelling of the Needham Schroeder Protocol.

Needham-Schroeder Protocol

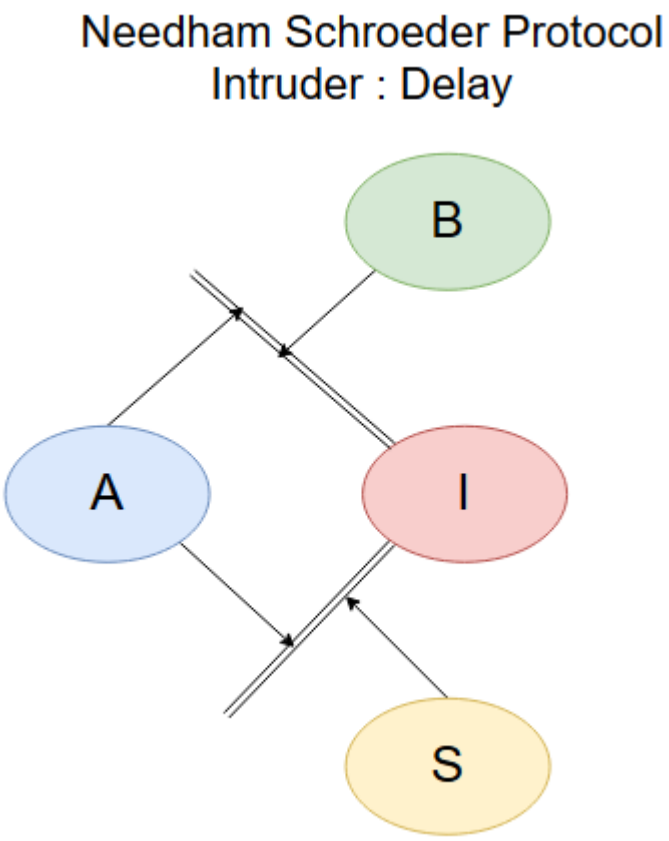
The classic implementation of the Needham Schroeder Protocol.



1.) A -> S : A,B,Na
2.) S -> A : (Na, B, Kab, {Kab, A}, Kb}Ka
3.) A -> B : {Kab, A}Kb
4.) B -> A :{Nb}Kab
5.) A -> B :{Nb-1}Kab

Intruder – Delay

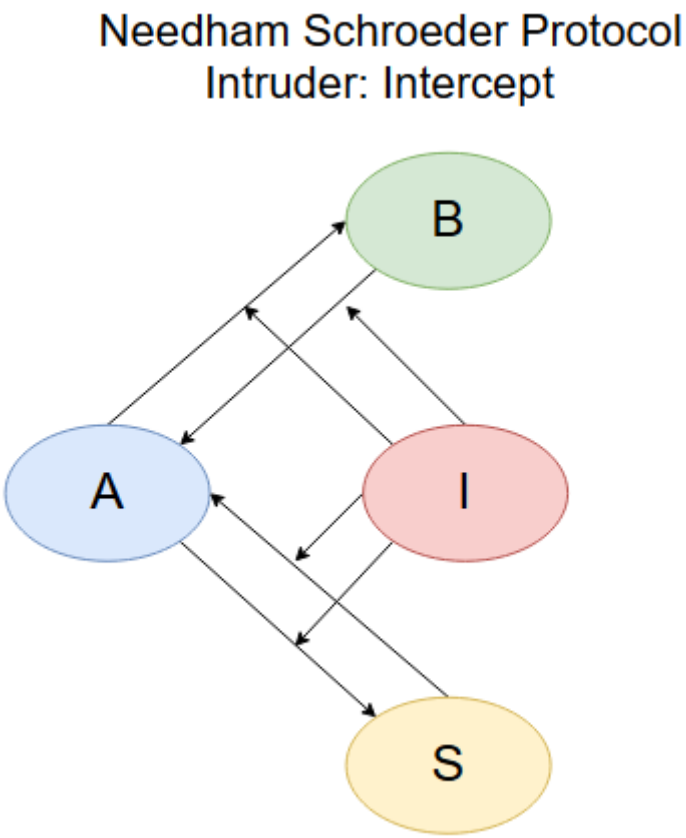
Intruder functionality, delaying messages transmitted along the message channel.



Intruder can delay for as long as it sees fit, including potentially indefinitely.

Intruder – Intercept

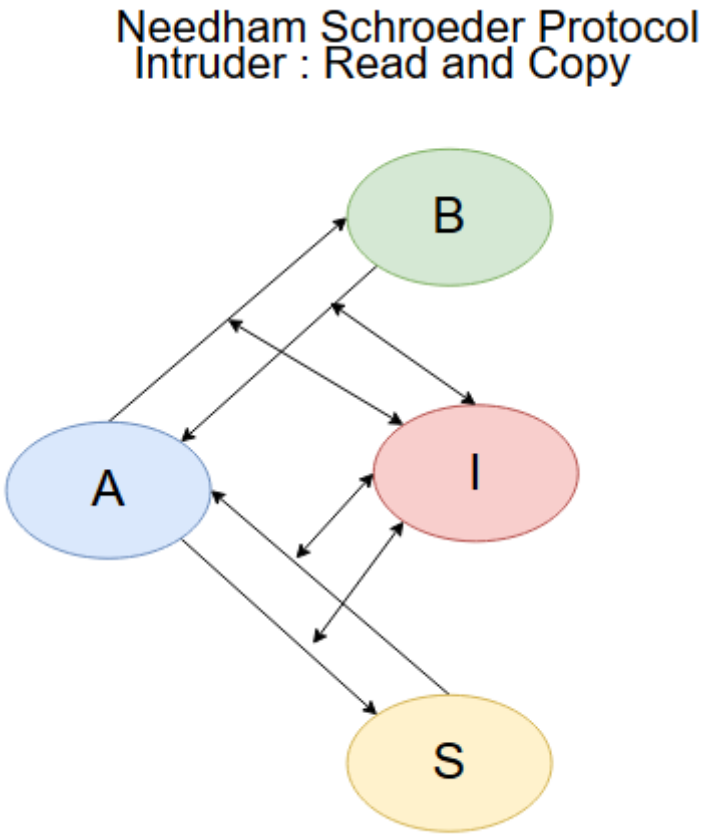
Intruder Functionality, intercept messages.



Unknown extent to functionality, if it stops messages then it is merely an infinite implementation of the delay, if it stops to read the messages then it is merely a variation of Read and Copy functionality.

Intruder – Read and Copy

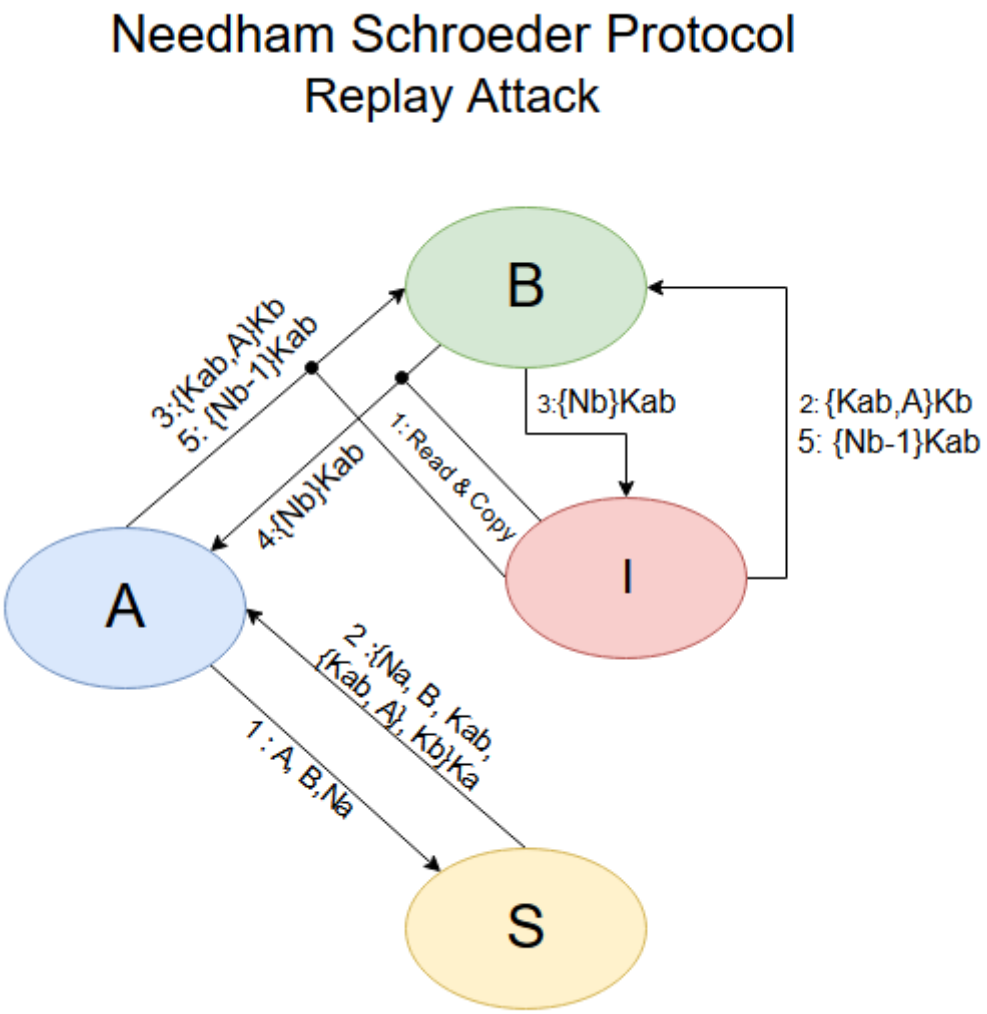
Intruder Functionality. Read the messages passed between the actors and store them in a memory buffer.



Intruder can only read the messages passed and has no ability to garner information on the variables within the actors, this means the only keys that are expressed by messages can be captured.

Intruder – Replay Attack

A classic vulnerability of the Needham-Schroeder protocol, utilizing an old version of Kab, the Intruder replays the {Kab, A}Kb message to B, who accepts it unable to tell it is an older compromised key.



This flaw is fixed in the Kerberos protocol by utilizing time stamps alongside keys, allowing the actors to identify if a key is fresh or not.

Verification and Testing

Testing the functionalities of the model, as well as its authentication and secrecy properties using CTL and LTL. The model is difficult to test with any specification that deems all cases must have the property, as the model is dependent on the passing of messages containing keys and therefore a single instance of delay being active is all it would take to falsify the specification, therefore most of the formulas will utilize specification for their existing a possible path.

Authentication

The central function of authentication regarding the Needham-Schroeder protocol is the passing of keys, each actor verifies each other by the possession of keys.

First, the verification that if allowed to progress normally there will exist a state where the machines A and B authenticate each other without fail. The formula essentially meaning that there exists a future where the two establish and verify each other's connections.

```
NuSMV > check_ctlspec -p "EF(mchA.bConnection = TRUE & mchB.aConnection = TRUE)"
-- specification EF (mchA.bConnection = TRUE & mchB.aConnection = TRUE)  is true
```

As aforementioned, this is not possible for all paths due to the existence of the intruder's delay functionality, where in there will always exist paths in which the intruder is able to delay the actors to such a degree that authentication is never established. As demonstrated below, a variation on the previous formula this time expressing that the actors authenticate in all futures.

```
NuSMV > check_ctlspec -p "AF(mchA.bConnection = TRUE & mchB.aConnection = TRUE)"
-- specification AF (mchA.bConnection = TRUE & mchB.aConnection = TRUE)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 8.1 <-
    mchA.nonceGen = FALSE
    mchA.aKey = nop
    mchA.authenticated = FALSE
    mchA.bConnection = FALSE
    serv.state = idle
    serv.sKey = nop
    mchB.bAuth = FALSE
    mchB.bKey = nop
    mchB.aConnection = FALSE
    mchB.iAuth = FALSE
    mchB.iConnection = FALSE
    int.delay = FALSE
    int.capture = FALSE
    int.capturedKey = nop
    int.replay = FALSE
    int.iKey = nop
    int.bConnection = FALSE
  -> State: 8.2 <-
    mchA.nonceGen = TRUE
  -> State: 8.3 <-
    mchA.aKey = a
  -> State: 8.4 <-
    serv.state = authing
    serv.sKey = a
  -> State: 8.5 <-
    mchA.authenticated = TRUE
  -> State: 8.6 <-
    mchA.aKey = b
  -> State: 8.7 <-
    serv.state = idle
    mchB.bAuth = TRUE
    mchB.bKey = b
  -> State: 8.8 <-
    mchB.bKey = ab
  -> State: 8.9 <-
    mchA.aKey = ab
  -- Loop starts here
  -> State: 8.10 <-
    mchB.aConnection = TRUE
    int.delay = TRUE
  -> State: 8.11 <-
```

This happens due to the assumption the intruder has total control over the communication channel, the intruder is able to delay messages indefinitely due to this power, one could technically consider this an erroneous result for testing the authentication process of the base Needham Schroeder model, as it relies on the assumption that after being delayed indefinitely, the actors would simply not bother authenticating utilising another secure channel.

We can continue to verify the authentication of machines utilizing the EF clause. Using this clause alongside eventualities that should definitely be false in order to express that authentication cannot occur unless everything works as intended. For example, if machine A is never authenticated by machine B, then the connection cannot exist:

```
NuSMV > check_ctlspec -p " EF(mchA.bAuth = FALSE & mchB.aConnection = TRUE)"
-- specification EF (mchA.bAuth = FALSE & mchB.aConnection = TRUE)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 9.1 <-
    mchA.nonceGen = FALSE
    mchA.aKey = nop
    mchA.authenticated = FALSE
    mchA.bConnection = FALSE
    serv.state = idle
    serv.sKey = nop
    mchB.bAuth = FALSE
    mchB.bKey = nop
    mchB.aConnection = FALSE
    mchB.iAuth = FALSE
    mchB.iConnection = FALSE
    int.delay = FALSE
    int.capture = FALSE
    int.capturedKey = nop
    int.replay = FALSE
    int.iKey = nop
    int.bConnection = FALSE
```

Furthermore, if machine A isn't authenticated by the server, then it cannot be authenticated by machine B:

```
NuSMV > check_ctlspec -p "EF(mchA.authenticated = FALSE & mchB.aConnection = TRUE)"
-- specification EF (mchA.authenticated = FALSE & mchB.aConnection = TRUE)  is false
-- as demonstrated by the following execution sequence
```

This is true for all paths:

```
NuSMV > check_ctlspec -p "AF(mchA.authenticated = FALSE & mchB.aConnection = TRUE)"
-- specification AF (mchA.authenticated = FALSE & mchB.aConnection = TRUE)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 6.1 <-
    mchA.nonceGen = FALSE
    mchA.aKey = nop
    mchA.authenticated = FALSE
    mchA.bConnection = FALSE
    serv.state = idle
    serv.sKey = nop
    mchB.bAuth = FALSE
    mchB.bKey = nop
    mchB.aConnection = FALSE
    mchB.iAuth = FALSE
    mchB.iConnection = FALSE
    int.delay = FALSE
    int.capture = FALSE
    int.capturedKey = nop
    int.replay = FALSE
    int.iKey = nop
    int.bConnection = FALSE
  -> State: 6.2 <-
    mchA.nonceGen = TRUE
  -> State: 6.3 <-
    mchA.aKey = a
  -> State: 6.4 <-
    serv.state = authing
    serv.sKey = a
  -> State: 6.5 <-
    mchA.authenticated = TRUE
  -> State: 6.6 <-
    mchA.aKey = b
  -> State: 6.7 <-
    serv.state = idle
    mchB.bAuth = TRUE
    mchB.bKey = b
  -> State: 6.8 <-
    mchB.bKey = ab
  -> State: 6.9 <-
    mchA.aKey = ab
  -> State: 6.10 <-
    mchB.aConnection = TRUE
  -- Loop starts here
  -> State: 6.11 <-
    mchA.bConnection = TRUE
  -> State: 6.12 <-
```

Secrecy

Regarding secrecy, we must verify that the intruder cannot always take action that leads to it establishing its own connection with machine B via replay attack. Despite this, since we have to assume that the intruder has complete control over the message channel, we cannot guarantee that in every single case the intruder does not read and copy messages, but we can test for scenarios where the intruder either doesn't read and copy, or perform a replay attack.

Firstly, testing if the intruder establishes connection with B in all paths:

```
NuSMV > check_ctlspec -p " AF(mchB.iConnection = TRUE & int.bConnection = TRUE)"
-- specification AF (mchB.iConnection = TRUE & int.bConnection = TRUE)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 32.1 <-
    mchA.nonceGen = FALSE
    mchA.aKey = nop
    mchA.authenticated = FALSE
    mchA.bConnection = FALSE
    serv.state = idle
    serv.sKey = nop
    mchB.bAuth = FALSE
    mchB.bKey = nop
    mchB.aConnection = FALSE
    mchB.iAuth = FALSE
    mchB.iConnection = FALSE
    int.delay = FALSE
    int.capture = FALSE
    int.capturedKey = nop
    int.replay = FALSE
    int.iKey = nop
    int.bConnection = FALSE
  -> State: 32.2 <-
    mchA.nonceGen = TRUE
  -> State: 32.3 <-
    mchA.aKey = a
  -> State: 32.4 <-
    serv.state = authing
    serv.sKey = a
  -> State: 32.5 <-
    mchA.authenticated = TRUE
  -> State: 32.6 <-
    mchA.aKey = b
  -> State: 32.7 <-
    serv.state = idle
    mchB.bAuth = TRUE
    mchB.bKey = b
  -> State: 32.8 <-
    mchB.bKey = ab
  -> State: 32.9 <-
    mchA.aKey = ab
  -> State: 32.10 <-
    mchB.aConnection = TRUE
  -- Loop starts here
  -> State: 32.11 <-
    mchA.bConnection = TRUE
  -> State: 32.12 <-
```

This counter example being the most prominent, a simple linear run of the Needham Schroeder protocol.

Secondly, testing if the intruder always steals the keys passed between actors:
(The counter examples have been left out to limit document length, they are the same as above)

```
NuSMV > check_ctlspec -p " AF(int.capturedKey = ab)"
-- specification AF int.capturedKey = ab  is false
-- as demonstrated by the following execution sequence
```

```
NuSMV > check_ctlspec -p " AF(int.capturedKey = b)"
-- specification AF int.capturedKey = b  is false
```

```
NuSMV > check_ctlspec -p " AF(int.capturedKey = a)"
-- specification AF int.capturedKey = a  is false
```

One could define this secrecy as being superficial however, due to the intruder having complete control one could assume that in these cases it simply didn't happen simply due to the intruder not wanting to rather than being unable to.

This is further compounded by that fact that there exist paths where the intruder can establish connection:

```
NuSMV > check_ctlspec -p " EF(mchB.iConnection = TRUE & int.bConnection = TRUE)"
-- specification EF (mchB.iConnection = TRUE & int.bConnection = TRUE)  is true
```

Then paths where the Intruder successfully reads and copies all the variations of keys:

```
NuSMV > check_ctlspec -p " EF(int.capturedKey = ab)"
-- specification EF int.capturedKey = ab  is true
```

```
NuSMV > check_ctlspec -p " EF(int.capturedKey = a)"
-- specification EF int.capturedKey = a  is true
```

```
NuSMV > check_ctlspec -p " EF(int.capturedKey = b)"
-- specification EF int.capturedKey = b  is true
```

In essence, so long as the intruder has the ability to capture messages and therefore breach keys at every pass, secrecy is never guaranteed, since all it takes is a single breached AB key to allow the intruder into the system via replay attack. Furthermore, given that the intruder is assumed to have complete control one could argue that there is no sense of secrecy at all as the intruder is always able to capture every message passed and therefore the potential of a replay attacks its always available. The intruder is always able to attack once it has seen a single key, it is not a matter of if, but when.

Shown below is the interaction of the possession of keys with the authentication process that take place between the intruder and machine B:

```
NuSMV > check_ctlspec -p "EF(int.capturedKey = ab & int.bConnection = TRUE)"
-- specification EF (int.capturedKey = ab & int.bConnection = TRUE)  is true
```

This is the expected setup and outcome of the replay attack.

```
NuSMV > check_ctlspec -p "EF(int.capturedKey = b & int.bConnection = TRUE)"
-- specification EF (int.capturedKey = b & int.bConnection = TRUE)  is true
```

This is true on a technicality, the key that the intruder has to pass to machine B is the B key, but the intruder cannot authenticate itself with just the B key.

```
-- specification EF (int.capturedKey = a & int.bConnection = TRUE)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 7.1 <-
    mchA.nonceGen = FALSE
    mchA.aKey = nop
    mchA.authenticated = FALSE
    mchA.bConnection = FALSE
    serv.state = idle
    serv.sKey = nop
    mchB.bAuth = FALSE
    mchB.bKey = nop
    mchB.aConnection = FALSE
    mchB.iAuth = FALSE
    mchB.iConnection = FALSE
    int.delay = FALSE
    int.capture = FALSE
    int.capturedKey = nop
    int.replay = FALSE
    int.iKey = nop
    int.bConnection = FALSE
```

Then finally the intruder can obviously not authenticate itself with machine B using the A key.

As expected, this is not always the case, with the “secure” run being the standard run of the Needham Schroeder protocol.

```
NuSMV > check_ctlspec -p "AF(int.iKey = ab & mchB.iConnection = TRUE)"
-- specification AF (int.iKey = ab & mchB.iConnection = TRUE)  is false
```

```
NuSMV > check_ctlspec -p "AF(int.iKey = b & mchB.iConnection = TRUE)"
-- specification AF (int.iKey = b & mchB.iConnection = TRUE)  is false
```

```
NuSMV > check_ctlspec -p "AF(int.iKey = a & mchB.iConnection = TRUE)"
-- specification AF (int.iKey = a & mchB.iConnection = TRUE)  is false
```

The traces have been deliberately left out to conserve document space, but they are all standard runs of the protocol.

Conclusion

The conclusion of this model, testing, and document overall is extremely simplistic. The Needham Schroeder protocol is not secure, at all. During testing it was frequently observed that the only time the model would run with a semblance of secrecy or genuine authenticity was when the intruder would do absolutely nothing. The power that the intruder holds in this implementation of the model gives it the ability to obliterate any and all authenticity and secrecy simply by acting once, even something as simple as delaying a message once and then doing nothing else removes the ability to guarantee authenticity and secrecy, all due to a act which could be considered a feint. One could even argue given this case that even in a textbook run of the Needham Schroeder protocol authenticity and secrecy cannot be guaranteed, especially in the case of this model. The intruder is always there, the clean run of the protocol is possible not because the intruder cannot do anything, but simply because it chooses not to.