HATS Deliverable D4.1 Appendix: Library of generated protocols

February 24, 2012

In the following we show a sample of protocols generated automatically.

The Taxpayer exmple

```
module TestProject;

import * from ABSProtocols;
import * from ExecutionEnvironment;

  def Bool testmessageFromSToCagentTPtextstringTAXIDTP (ProtocolClause msg)
= case msg {Message( AgentVar( "S" ),    AgentVar( "C" ),

       Cons (   Agent(AgentVar( "TP" )),
       Cons (   TextOrigin( "TAXID" , AgentVar( "TP" )) ,   Nil
  )  )  )      => True ;

     _ => False ;

     }  ;


  def Bool testmessageFromSToCtextstringCHARITYPAYMENTTP (ProtocolClause msg)
= case msg {Message( AgentVar( "S" ),    AgentVar( "C" ),

       Cons (   TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" )) ,   Nil
  )  )      => True ;

     _ => False ;

     }  ;



class  Cclass (AgentTerm name, Network network) implements Agent{
Bool messageFromSToCagentTPtextstringTAXIDTPQuery = False  ;

   Bool messageFromSToCtextstringCHARITYPAYMENTTPQuery = False  ;

    ProtocolClause messageFromSToCagentTPtextstringTAXIDTP =  UndefinedClause   ;

    PayloadElement agentS =  UndefinedElem   ;

    PayloadElement agentTP =  UndefinedElem   ;

    PayloadElement textstringTAXIDTP =  UndefinedElem   ;

    ProtocolClause messageFromSToCtextstringCHARITYPAYMENTTP =  UndefinedClause   ;

    PayloadElement textstringCHARITYPAYMENTTP =  UndefinedElem   ;

      Unit run(){ await(network != null) ;

    messageFromSToCagentTPtextstringTAXIDTP = Message(AgentVar( "S" ),AgentVar( "C" ),
 list[  Agent(AgentVar( "TP" )),  TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ;

   await( messageFromSToCagentTPtextstringTAXIDTPQuery )  ;

   agentS =  Agent(AgentVar( "S" )) ;

   agentTP =  Agent(AgentVar( "TP" )) ;

   textstringTAXIDTP =  TextOrigin( "TAXID" , AgentVar( "TP" ))  ;

   messageFromSToCtextstringCHARITYPAYMENTTP = Message(AgentVar( "S" ),AgentVar( "C" ),
 list[  TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" )) ]) ;

   await( messageFromSToCtextstringCHARITYPAYMENTTPQuery )  ;

   textstringCHARITYPAYMENTTP =  TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" ))  ;
```

```
    }
  /* End of run-method */


   Unit receive(ProtocolClause msg){
 if ( testmessageFromSToCagentTPtextstringTAXIDTP (msg) )
 {messageFromSToCagentTPtextstringTAXIDTPQuery = True ;

    }
    else  {
 if ( testmessageFromSToCtextstringCHARITYPAYMENTTP (msg) )
 {messageFromSToCtextstringCHARITYPAYMENTTPQuery = True ;

    }
    else  { skip  ;

      }
  /* End of test method */

    }
  /* End of test method */

 }
  /* End of receive method */

 }
 /* End of class */

 def Bool testmessageFromTPToStextstringTAXIDTPtextRequestGetTaxes (ProtocolClause msg)
= case msg {Message( AgentVar( "TP" ),   AgentVar( "S" ),

      Cons (  TextOrigin( "TAXID" , AgentVar( "TP" )) ,
      Cons ( Text("Request_Get_Taxes"),  Nil
 ) ) ) )     => True ;

    _ => False ;

    }  ;


 def Bool
  testmessageFromTPToStextstringTAXPAYMENTTPtextstringCHARITYPAYMENTTP (ProtocolClause msg)
= case msg {Message( AgentVar( "TP" ),   AgentVar( "S" ),

      Cons (  TextOrigin( "TAXPAYMENT" , AgentVar( "TP" )) ,
      Cons (  TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" )) ,  Nil
 ) ) ) )     => True ;

    _ => False ;

    }  ;


 def Bool testmessageFromTCToStextValidatetextstringTAXIDTP (ProtocolClause msg)
= case msg {Message( AgentVar( "TC" ),   AgentVar( "S" ),

      Cons ( Text("Validate"),
      Cons (  TextOrigin( "TAXID" , AgentVar( "TP" )) ,  Nil
 ) ) ) )     => True ;

    _ => False ;

    }  ;


 def Bool testmessageFromTPToStextBooleanResult (ProtocolClause msg)
= case msg {Message( AgentVar( "TP" ),   AgentVar( "S" ),

      Cons ( Text("Boolean_Result"),  Nil
 ) )     => True ;

    _ => False ;
```

```
    } ;


 def Bool testmessageFromTCToStextFreezetextstringTAXIDTP (ProtocolClause msg)
= case msg {Message( AgentVar( "TC" ),    AgentVar( "S" ),

       Cons ( Text("Freeze"),
       Cons (  TextOrigin( "TAXID" , AgentVar( "TP" )) ,  Nil
 ) ) )      => True ;

    _ => False ;

    } ;



class  Sclass (AgentTerm name, Network network) implements Agent{
Bool messageFromTPToStextstringTAXIDTPtextRequestGetTaxesQuery = False  ;

  Bool messageFromTPToStextstringTAXPAYMENTTPtextstringCHARITYPAYMENTTPQuery = False  ;

  Bool messageFromTCToStextValidatetextstringTAXIDTPQuery = False  ;

  Bool messageFromTPToStextBooleanResultQuery = False  ;

  Bool messageFromTCToStextFreezetextstringTAXIDTPQuery = False  ;

   PayloadElement agentTP =  UndefinedElem   ;

   PayloadElement textstringTAXIDTP =  UndefinedElem   ;

   ProtocolClause messageFromSToTPagentTPtextstringTAXIDTP =  UndefinedClause  ;

   PayloadElement agentTC =  UndefinedElem   ;

   ProtocolClause messageFromSToTCagentTPtextstringTAXIDTP =  UndefinedClause  ;

   PayloadElement agentC =  UndefinedElem   ;

   ProtocolClause messageFromSToCagentTPtextstringTAXIDTP =  UndefinedClause  ;

   ProtocolClause
     messageFromTPToStextstringTAXIDTPtextRequestGetTaxes =  UndefinedClause   ;

   PayloadElement textRequestGetTaxes =  UndefinedElem   ;

   PayloadElement textstringAMOUNTS =  UndefinedElem   ;

   ProtocolClause messageFromSToTPtextstringAMOUNTS =  UndefinedClause  ;

   ProtocolClause
      messageFromTPToStextstringTAXPAYMENTTPtextstringCHARITYPAYMENTTP =
         UndefinedClause   ;

   PayloadElement textstringTAXPAYMENTTP =  UndefinedElem   ;

   PayloadElement textstringCHARITYPAYMENTTP =  UndefinedElem   ;

   PayloadElement textCheckTax =  UndefinedElem   ;

   ProtocolClause messageFromSToTCtextCheckTax =  UndefinedClause  ;

   ProtocolClause messageFromTCToStextValidatetextstringTAXIDTP =  UndefinedClause   ;

   PayloadElement textValidate =  UndefinedElem   ;

   ProtocolClause messageFromTPToStextBooleanResult =  UndefinedClause   ;

   PayloadElement textBooleanResult =  UndefinedElem   ;
```

```
    ProtocolClause messageFromTCToStextFreezetextstringTAXIDTP = UndefinedClause ;

    PayloadElement textFreeze = UndefinedElem ;

    ProtocolClause messageFromSToCtextstringCHARITYPAYMENTTP = UndefinedClause ;

      Unit run(){ await(network != null) ;

     agentTP = Agent(AgentVar( "TP" )) ;

   textstringTAXIDTP = TextOrigin( "TAXID" , AgentVar( "TP" )) ;

   messageFromSToTPagentTPtextstringTAXIDTP = Message(AgentVar( "S" ),AgentVar( "TP" ),
  list[ Agent(AgentVar( "TP" )), TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ;

   network!send( Message(AgentVar( "S" ),AgentVar( "TP" ),
  list[ Agent(AgentVar( "TP" )), TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ) ;

   agentTC = Agent(AgentVar( "TC" )) ;

   messageFromSToTCagentTPtextstringTAXIDTP = Message(AgentVar( "S" ),AgentVar( "TC" ),
  list[ Agent(AgentVar( "TP" )), TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ;

   network!send( Message(AgentVar( "S" ),AgentVar( "TC" ),
  list[ Agent(AgentVar( "TP" )), TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ) ;

   agentC = Agent(AgentVar( "C" )) ;

   messageFromSToCagentTPtextstringTAXIDTP = Message(AgentVar( "S" ),AgentVar( "C" ),
  list[ Agent(AgentVar( "TP" )), TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ;

   network!send( Message(AgentVar( "S" ),AgentVar( "C" ),
  list[ Agent(AgentVar( "TP" )), TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ) ;

   messageFromTPToStextstringTAXIDTPtextRequestGetTaxes =
 Message(AgentVar( "TP" ),AgentVar( "S" ),
  list[ TextOrigin( "TAXID" , AgentVar( "TP" )) , Text("Request␣Get␣Taxes")]) ;

   await( messageFromTPToStextstringTAXIDTPtextRequestGetTaxesQuery ) ;

   textRequestGetTaxes = Text("Request␣Get␣Taxes") ;

   textstringAMOUNTS = TextOrigin( "AMOUNT" , AgentVar( "S" )) ;

   messageFromSToTPtextstringAMOUNTS = Message(AgentVar( "S" ),AgentVar( "TP" ),
  list[ TextOrigin( "AMOUNT" , AgentVar( "S" )) ]) ;

   network!send( Message(AgentVar( "S" ),AgentVar( "TP" ),
  list[ TextOrigin( "AMOUNT" , AgentVar( "S" )) ]) ) ;

   messageFromTPToStextstringTAXPAYMENTTPtextstringCHARITYPAYMENTTP =
Message(AgentVar( "TP" ),AgentVar( "S" ),
  list[ TextOrigin( "TAXPAYMENT" , AgentVar( "TP" )) ,
        TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" )) ]) ;

   await( messageFromTPToStextstringTAXPAYMENTTPtextstringCHARITYPAYMENTTPQuery ) ;

   textstringTAXPAYMENTTP = TextOrigin( "TAXPAYMENT" , AgentVar( "TP" )) ;

   textstringCHARITYPAYMENTTP = TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" )) ;

   textCheckTax = Text("Check␣Tax") ;

   messageFromSToTCtextCheckTax = Message(AgentVar( "S" ),AgentVar( "TC" ),
  list[ Text("Check␣Tax")]) ;

   network!send( Message(AgentVar( "S" ),AgentVar( "TC" ),
  list[ Text("Check␣Tax")]) ) ;

   messageFromTCToStextValidatetextstringTAXIDTP = Message(AgentVar( "TC" ),AgentVar( "S" ),
  list[ Text("Validate"), TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ;
```

```
 await( messageFromTCToStextValidatetextstringTAXIDTPQuery )  ;

 textValidate = Text("Validate") ;

 messageFromTPToStextBooleanResult = Message(AgentVar( "TP" ),AgentVar( "S" ),
list[ Text("Boolean␣Result")]) ;

 await( messageFromTPToStextBooleanResultQuery )  ;

 textBooleanResult = Text("Boolean␣Result") ;

 messageFromTCToStextFreezetextstringTAXIDTP = Message(AgentVar( "TC" ),AgentVar( "S" ),
list[ Text("Freeze"),  TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ;

 await( messageFromTCToStextFreezetextstringTAXIDTPQuery )  ;

 textFreeze = Text("Freeze") ;

 messageFromSToCtextstringCHARITYPAYMENTTP = Message(AgentVar( "S" ),AgentVar( "C" ),
list[  TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" )) ]) ;

 network!send( Message(AgentVar( "S" ),AgentVar( "C" ),
list[  TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" )) ]) )  ;

   }
 /* End of run-method */


  Unit receive(ProtocolClause msg){
if ( testmessageFromTPToStextstringTAXIDTPtextRequestGetTaxes (msg) )
{messageFromTPToStextstringTAXIDTPtextRequestGetTaxesQuery = True ;

   }
   else  {
if ( testmessageFromTPToStextstringTAXPAYMENTTPtextstringCHARITYPAYMENTTP (msg) )
{messageFromTPToStextstringTAXPAYMENTTPtextstringCHARITYPAYMENTTPQuery = True ;

   }
   else  {
if ( testmessageFromTCToStextValidatetextstringTAXIDTP (msg) )
{messageFromTCToStextValidatetextstringTAXIDTPQuery = True ;

   }
   else  {
if ( testmessageFromTPToStextBooleanResult (msg) )
{messageFromTPToStextBooleanResultQuery = True ;

   }
   else  {
if ( testmessageFromTCToStextFreezetextstringTAXIDTP (msg) )
{messageFromTCToStextFreezetextstringTAXIDTPQuery = True ;

   }
   else  { skip  ;

     }
 /* End of test method */

   }
 /* End of test method */

   }
 /* End of test method */

   }
 /* End of test method */

   }
 /* End of test method */

}
 /* End of receive method */
```

```
 }
 /* End of class */

 def Bool testmessageFromSToTCagentTPtextstringTAXIDTP (ProtocolClause msg)
= case msg {Message( AgentVar( "S" ),    AgentVar( "TC" ),

       Cons (   Agent(AgentVar( "TP" )),
       Cons (   TextOrigin( "TAXID" , AgentVar( "TP" )) ,   Nil
 )  )  )      => True ;

    _ => False ;

    }  ;


 def Bool testmessageFromSToTCtextCheckTax (ProtocolClause msg)
= case msg {Message( AgentVar( "S" ),    AgentVar( "TC" ),

       Cons ( Text("Check Tax"),   Nil
 )  )      => True ;

    _ => False ;

    }  ;



class  TCclass (AgentTerm name, Network network) implements Agent{
Bool messageFromSToTCagentTPtextstringTAXIDTPQuery = False  ;

  Bool messageFromSToTCtextCheckTaxQuery = False  ;

   ProtocolClause messageFromSToTCagentTPtextstringTAXIDTP =  UndefinedClause   ;

   PayloadElement agentS =  UndefinedElem   ;

   PayloadElement agentTP =  UndefinedElem   ;

   PayloadElement textstringTAXIDTP =  UndefinedElem   ;

   ProtocolClause messageFromSToTCtextCheckTax =  UndefinedClause   ;

   PayloadElement textCheckTax =  UndefinedElem   ;

   PayloadElement textValidate =  UndefinedElem   ;

   ProtocolClause messageFromTCToStextValidatetextstringTAXIDTP =  UndefinedClause  ;

   PayloadElement textFreeze =  UndefinedElem   ;

   ProtocolClause messageFromTCToStextFreezetextstringTAXIDTP =  UndefinedClause  ;

     Unit run(){ await(network != null) ;

    messageFromSToTCagentTPtextstringTAXIDTP = Message(AgentVar( "S" ),AgentVar( "TC" ),
 list[  Agent(AgentVar( "TP" )),  TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ;

  await( messageFromSToTCagentTPtextstringTAXIDTPQuery )  ;

  agentS =  Agent(AgentVar( "S" )) ;

  agentTP =  Agent(AgentVar( "TP" )) ;

  textstringTAXIDTP =  TextOrigin( "TAXID" , AgentVar( "TP" ))  ;

  messageFromSToTCtextCheckTax = Message(AgentVar( "S" ),AgentVar( "TC" ),
 list[ Text("Check Tax")]) ;

  await( messageFromSToTCtextCheckTaxQuery )  ;

  textCheckTax = Text("Check Tax") ;
```

```
  textValidate = Text("Validate") ;

  messageFromTCToStextValidatetextstringTAXIDTP = Message(AgentVar( "TC" ),AgentVar( "S" ),
 list[ Text("Validate"),  TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ;

  network!send( Message(AgentVar( "TC" ),AgentVar( "S" ),
 list[ Text("Validate"),  TextOrigin( "TAXID" , AgentVar( "TP" )) ]) )  ;

  textFreeze = Text("Freeze") ;

  messageFromTCToStextFreezetextstringTAXIDTP = Message(AgentVar( "TC" ),AgentVar( "S" ),
 list[ Text("Freeze"),  TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ;

  network!send( Message(AgentVar( "TC" ),AgentVar( "S" ),
 list[ Text("Freeze"),  TextOrigin( "TAXID" , AgentVar( "TP" )) ]) )  ;

    }
  /* End of run-method */


   Unit receive(ProtocolClause msg){
 if ( testmessageFromSToTCagentTPtextstringTAXIDTP (msg) )
 {messageFromSToTCagentTPtextstringTAXIDTPQuery = True ;

    }
    else  {
 if ( testmessageFromSToTCtextCheckTax (msg) )
 {messageFromSToTCtextCheckTaxQuery = True ;

    }
    else  { skip  ;

      }
  /* End of test method */

    }
  /* End of test method */

 }
  /* End of receive method */

 }
 /* End of class */

 def Bool testmessageFromSToTPagentTPtextstringTAXIDTP (ProtocolClause msg)
= case msg {Message( AgentVar( "S" ),   AgentVar( "TP" ),

      Cons (  Agent(AgentVar( "TP" )),
      Cons (  TextOrigin( "TAXID" , AgentVar( "TP" )) ,  Nil
 ) ) )     => True ;

    _ => False ;

    }  ;


 def Bool testmessageFromSToTPtextstringAMOUNTS (ProtocolClause msg)
= case msg {Message( AgentVar( "S" ),   AgentVar( "TP" ),

      Cons (  TextOrigin( "AMOUNT" , AgentVar( "S" )) ,  Nil
 ) )     => True ;

    _ => False ;

    }  ;



class  TPclass (AgentTerm name, Network network) implements Agent{
Bool messageFromSToTPagentTPtextstringTAXIDTPQuery = False  ;
```

```
  Bool messageFromSToTPtextstringAMOUNTSQuery = False  ;

   ProtocolClause messageFromSToTPagentTPtextstringTAXIDTP =  UndefinedClause   ;

   PayloadElement agentS =  UndefinedElem   ;

   PayloadElement agentTP =  UndefinedElem   ;

   PayloadElement textstringTAXIDTP =  UndefinedElem   ;

   PayloadElement textRequestGetTaxes =  UndefinedElem   ;

   ProtocolClause messageFromTPToStextstringTAXIDTPtextRequestGetTaxes =  UndefinedClause  ;

   ProtocolClause messageFromSToTPtextstringAMOUNTS =  UndefinedClause   ;

   PayloadElement textstringAMOUNTS =  UndefinedElem   ;

   PayloadElement textstringTAXPAYMENTTP =  UndefinedElem   ;

   PayloadElement textstringCHARITYPAYMENTTP =  UndefinedElem   ;

   ProtocolClause
messageFromTPToStextstringTAXPAYMENTTPtextstringCHARITYPAYMENTTP =  UndefinedClause  ;

   PayloadElement textBooleanResult =  UndefinedElem   ;

   ProtocolClause messageFromTPToStextBooleanResult =  UndefinedClause  ;

     Unit run(){ await(network != null) ;

    messageFromSToTPagentTPtextstringTAXIDTP =
Message(AgentVar( "S" ),AgentVar( "TP" ),
 list[  Agent(AgentVar( "TP" )),  TextOrigin( "TAXID" , AgentVar( "TP" )) ]) ;

  await( messageFromSToTPagentTPtextstringTAXIDTPQuery )  ;

  agentS =  Agent(AgentVar( "S" )) ;

  agentTP =  Agent(AgentVar( "TP" )) ;

  textstringTAXIDTP =  TextOrigin( "TAXID" , AgentVar( "TP" ))  ;

  textRequestGetTaxes = Text("Request Get Taxes") ;

  messageFromTPToStextstringTAXIDTPtextRequestGetTaxes =
Message(AgentVar( "TP" ),AgentVar( "S" ),
 list[  TextOrigin( "TAXID" , AgentVar( "TP" )) , Text("Request Get Taxes")]) ;

  network!send( Message(AgentVar( "TP" ),AgentVar( "S" ),
 list[  TextOrigin( "TAXID" , AgentVar( "TP" )) , Text("Request Get Taxes")]) )  ;

  messageFromSToTPtextstringAMOUNTS = Message(AgentVar( "S" ),AgentVar( "TP" ),
 list[  TextOrigin( "AMOUNT" , AgentVar( "S" )) ]) ;

  await( messageFromSToTPtextstringAMOUNTSQuery )  ;

  textstringAMOUNTS =  TextOrigin( "AMOUNT" , AgentVar( "S" ))  ;

  textstringTAXPAYMENTTP =  TextOrigin( "TAXPAYMENT" , AgentVar( "TP" ))  ;

  textstringCHARITYPAYMENTTP =  TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" ))  ;

  messageFromTPToStextstringTAXPAYMENTTPtextstringCHARITYPAYMENTTP =
    Message(AgentVar( "TP" ),AgentVar( "S" ),
 list[  TextOrigin( "TAXPAYMENT" , AgentVar( "TP" )) ,
           TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" )) ]) ;

  network!send( Message(AgentVar( "TP" ),AgentVar( "S" ),
 list[  TextOrigin( "TAXPAYMENT" , AgentVar( "TP" )) ,
           TextOrigin( "CHARITYPAYMENT" , AgentVar( "TP" )) ]) )  ;
```

```
  textBooleanResult = Text("Boolean Result") ;

  messageFromTPToStextBooleanResult = Message(AgentVar( "TP" ),AgentVar( "S" ),
 list[ Text("Boolean Result")]) ;

  network!send( Message(AgentVar( "TP" ),AgentVar( "S" ),
 list[ Text("Boolean Result")]) )  ;

    }
  /* End of run-method */


   Unit receive(ProtocolClause msg){
 if ( testmessageFromSToTPagentTPtextstringTAXIDTP (msg) )
 {messageFromSToTPagentTPtextstringTAXIDTPQuery = True ;

    }
    else  {
 if ( testmessageFromSToTPtextstringAMOUNTS (msg) )
 {messageFromSToTPtextstringAMOUNTSQuery = True ;

    }
    else  { skip  ;

      }
  /* End of test method */

    }
  /* End of test method */

 }
  /* End of receive method */

 }
 /* End of class */



// The protocol: TaxPayment Protocol
{
Agent zombieAgent =  new cog ZombieAgent(AgentName("TestZombie") )  ;

  Network network  ;

  network =  new cog Network(zombieAgent)  ;

  Agent objectTP ;

  objectTP = new cog TPclass (AgentVar( "TP" ) , network) ;

  network!register(AgentVar( "TP" ), objectTP)  ;

  Agent objectTC ;

  objectTC = new cog TCclass (AgentVar( "TC" ) , network) ;

  network!register(AgentVar( "TC" ), objectTC)  ;

  Agent objectC ;

  objectC = new cog Cclass (AgentVar( "C" ) , network) ;

  network!register(AgentVar( "C" ), objectC)  ;

  Agent objectS ;

  objectS = new cog Sclass (AgentVar( "S" ) , network) ;

  network!register(AgentVar( "S" ), objectS)  ;

  }
```

```
// end of file
```

ISO One pass unilateral

```
module TestProject;

import * from ABSProtocols;
import * from ExecutionEnvironment;


 class  Aclass (AgentTerm name, Network network) implements Agent{
  PayloadElement textText2A =  UndefinedElem    ;

    PayloadElement newnonceNAA =  UndefinedElem    ;

    PayloadElement agentB =  UndefinedElem    ;

    PayloadElement textText1A =  UndefinedElem    ;

    PayloadElement keyKeyAB =  UndefinedElem    ;

    PayloadElement encryptKeyABnonceNAAagentBtextText1A =  UndefinedElem    ;

    ProtocolClause messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A  =
            UndefinedClause;

      Unit run(){ await(network != null) ;

     textText2A =  TextOrigin("Text2", AgentVar( "A" ))   ;

   newnonceNAA =  New(Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ) ) ;

   agentB =  Agent(AgentVar( "B" )) ;

   textText1A =  TextOrigin("Text1", AgentVar( "A" ))   ;

   keyKeyAB =  Key(KeySymmetric( AgentVar( "A" ),AgentVar( "B" ))) ;

   encryptKeyABnonceNAAagentBtextText1A =
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
   list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
            Agent(AgentVar( "B" )), TextOrigin("Text1", AgentVar( "A" )) ] ) ;

   messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A =

Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text2", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
 list[ Nonce( NonceVar ( "NA" ),
       AgentVar( "A" ) ),  Agent(AgentVar( "B" )), TextOrigin("Text1", AgentVar( "A" )) ] )]) ;

   network!send( Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text2", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
 list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
           Agent(AgentVar( "B" )),  TextOrigin("Text1", AgentVar( "A" )) ] )]) )  ;

     }
   /* End of run-method */


    Unit receive(ProtocolClause msg){skip  ;

    }
   /* End of receive method */

  }
  /* End of class */

  def Bool
   testmessageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A (ProtocolClause msg)
   = case msg {Message( AgentVar( "A" ),   AgentVar( "B" ),
```

```
        Cons (  TextOrigin("Text2", AgentVar( "A" )) ,
        Cons (  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
        Cons ( Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
        Cons (  Agent(AgentVar( "B" )),
        Cons (  TextOrigin("Text1", AgentVar( "A" )) ,  Nil
  ) ) ) ) ) ,  Nil
  ) ) ) )     => True ;

     _ => False ;

     } ;



 class  Bclass (AgentTerm name, Network network) implements Agent{
 Bool messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1AQuery = False  ;

     ProtocolClause messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A =
        UndefinedClause ;

     PayloadElement agentA =  UndefinedElem   ;

     PayloadElement textText2A =  UndefinedElem   ;

     PayloadElement keyKeyAB =  UndefinedElem   ;

     PayloadElement decryptKeyABencryptKeyABnonceNAAagentBtextText1A =  UndefinedElem   ;

     PayloadElement nonceNAA =  UndefinedElem   ;

     PayloadElement agentB =  UndefinedElem   ;

     PayloadElement textText1A =  UndefinedElem   ;

       Unit run(){ await(network != null) ;

      messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A =

Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text2", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
  list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
       Agent(AgentVar( "B" )),  TextOrigin("Text1", AgentVar( "A" )) ] )]) ;

    await( messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1AQuery )  ;

    agentA =  Agent(AgentVar( "A" )) ;

    textText2A =  TextOrigin("Text2", AgentVar( "A" ))  ;

    keyKeyAB =  Key(KeySymmetric( AgentVar( "A" ),AgentVar( "B" ))) ;

    decryptKeyABencryptKeyABnonceNAAagentBtextText1A =
  Decrypt(  KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
  list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
        Agent(AgentVar( "B" )),  TextOrigin("Text1", AgentVar( "A" )) ] ) ] ) ;

    nonceNAA = Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ) ;

    agentB =  Agent(AgentVar( "B" )) ;

    textText1A =  TextOrigin("Text1", AgentVar( "A" ))  ;

      }
    /* End of run-method */


     Unit receive(ProtocolClause msg){
  if ( testmessageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A (msg) )
```

```
 {messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1AQuery = True ;

    }
    else { skip ;

      }
  /* End of test method */

 }
  /* End of receive method */

 }
 /* End of class */


// The protocol: ISO-SYM-One-Pass-unilateral Authentication
{
Agent zombieAgent = new cog ZombieAgent(AgentName("TestZombie")) ;

  Network network ;

  network = new cog Network(zombieAgent) ;

  Agent objectA ;

  objectA = new cog Aclass (AgentVar( "A" ) , network) ;

  network!register(AgentVar( "A" ), objectA) ;

  Agent objectB ;

  objectB = new cog Bclass (AgentVar( "B" ) , network) ;

  network!register(AgentVar( "B" ), objectB) ;

  }
```

ISO One two-pass unilateral

```
 module TestProject;

import * from ABSProtocols;
import * from ExecutionEnvironment;

  def Bool testmessageFromBToAnonceNBBtextText1B (ProtocolClause msg)
 = case msg {Message( AgentVar( "B" ),   AgentVar( "A" ),

       Cons ( Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
       Cons (  TextOrigin("Text1", AgentVar( "B" )) ,  Nil
  ) ) )     => True ;

     _ => False ;

     } ;


 class  Aclass (AgentTerm name, Network network) implements Agent{
 Bool messageFromBToAnonceNBBtextText1BQuery = False  ;

    ProtocolClause messageFromBToAnonceNBBtextText1B =  UndefinedClause   ;

    PayloadElement agentB =  UndefinedElem   ;

    PayloadElement nonceNBB =  UndefinedElem   ;

    PayloadElement textText1B =  UndefinedElem   ;

    PayloadElement textText3A =  UndefinedElem   ;

    PayloadElement textText2A =  UndefinedElem   ;
```

```
    PayloadElement keyKeyAB = UndefinedElem ;

    PayloadElement encryptKeyABnonceNBBagentBtextText2A = UndefinedElem ;

    ProtocolClause
     messageFromAToBtextText3AencryptKeyABnonceNBBagentBtextText2A = UndefinedClause ;

      Unit run(){ await(network != null) ;

     messageFromBToAnonceNBBtextText1B =
 Message(AgentVar( "B" ),AgentVar( "A" ),
  list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ), TextOrigin("Text1", AgentVar( "B" )) ]) ;

   await( messageFromBToAnonceNBBtextText1BQuery ) ;

   agentB = Agent(AgentVar( "B" )) ;

   nonceNBB = Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ) ;

   textText1B = TextOrigin("Text1", AgentVar( "B" )) ;

   textText3A = TextOrigin("Text3", AgentVar( "A" )) ;

   textText2A = TextOrigin("Text2", AgentVar( "A" )) ;

   keyKeyAB = Key(KeySymmetric( AgentVar( "A" ),AgentVar( "B" ))) ;

   encryptKeyABnonceNBBagentBtextText2A =
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
      Agent(AgentVar( "B" )), TextOrigin("Text2", AgentVar( "A" )) ] ) ;

   messageFromAToBtextText3AencryptKeyABnonceNBBagentBtextText2A =
        Message(AgentVar( "A" ),AgentVar( "B" ),
  list[ TextOrigin("Text3", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
       list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
             Agent(AgentVar( "B" )), TextOrigin("Text2", AgentVar( "A" )) ] )]) ;

   network!send( Message(AgentVar( "A" ),AgentVar( "B" ),
  list[ TextOrigin("Text3", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
        list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
              Agent(AgentVar( "B" )), TextOrigin("Text2", AgentVar( "A" )) ] )]) ) ;

    }
  /* End of run-method */


    Unit receive(ProtocolClause msg){
  if ( testmessageFromBToAnonceNBBtextText1B (msg) )
  {messageFromBToAnonceNBBtextText1BQuery = True ;

    }
    else { skip ;

      }
  /* End of test method */

  }
   /* End of receive method */

  }
  /* End of class */

  def Bool testmessageFromAToBtextText3AencryptKeyABnonceNBBagentBtextText2A (ProtocolClause msg)
 = case msg {Message( AgentVar( "A" ), AgentVar( "B" ),

       Cons ( TextOrigin("Text3", AgentVar( "A" )) ,
       Cons ( Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
       Cons ( Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
       Cons ( Agent(AgentVar( "B" )),
```

```
      Cons ( TextOrigin("Text2", AgentVar( "A" )) , Nil
) ) ) ) , Nil
) ) ) => True ;

  _ => False ;

  } ;


class Bclass (AgentTerm name, Network network) implements Agent{
Bool messageFromAToBtextText3AencryptKeyABnonceNBBagentBtextText2AQuery = False ;

  PayloadElement newnonceNBB = UndefinedElem ;

  PayloadElement textText1B = UndefinedElem ;

  PayloadElement agentA = UndefinedElem ;

  ProtocolClause messageFromBToAnonceNBBtextText1B = UndefinedClause ;

  ProtocolClause messageFromAToBtextText3AencryptKeyABnonceNBBagentBtextText2A =
                 UndefinedClause ;

  PayloadElement textText3A = UndefinedElem ;

  PayloadElement keyKeyAB = UndefinedElem ;

  PayloadElement decryptKeyABencryptKeyABnonceNBBagentBtextText2A = UndefinedElem ;

  PayloadElement agentB = UndefinedElem ;

  PayloadElement textText2A = UndefinedElem ;

    Unit run(){ await(network != null) ;

   newnonceNBB = New(Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ) ) ;

 textText1B = TextOrigin("Text1", AgentVar( "B" )) ;

 agentA = Agent(AgentVar( "A" )) ;

 messageFromBToAnonceNBBtextText1B = Message(AgentVar( "B" ),AgentVar( "A" ),
list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ), TextOrigin("Text1", AgentVar( "B" )) ]) ;

 network!send( Message(AgentVar( "B" ),AgentVar( "A" ),
list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ), TextOrigin("Text1", AgentVar( "B" )) ]) ) ;

 messageFromAToBtextText3AencryptKeyABnonceNBBagentBtextText2A =
  Message(AgentVar( "A" ),AgentVar( "B" ),
list[ TextOrigin("Text3", AgentVar( "A" )) ,
Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
 list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
       Agent(AgentVar( "B" )), TextOrigin("Text2", AgentVar( "A" )) ] )]) ;

 await( messageFromAToBtextText3AencryptKeyABnonceNBBagentBtextText2AQuery ) ;

 textText3A = TextOrigin("Text3", AgentVar( "A" )) ;

 keyKeyAB = Key(KeySymmetric( AgentVar( "A" ),AgentVar( "B" ))) ;

 decryptKeyABencryptKeyABnonceNBBagentBtextText2A =
Decrypt( KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
  list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
        Agent(AgentVar( "B" )), TextOrigin("Text2", AgentVar( "A" )) ] ) ] ) ;

 agentB = Agent(AgentVar( "B" )) ;

 textText2A = TextOrigin("Text2", AgentVar( "A" )) ;

  }
```

```
   /* End of run-method */


    Unit receive(ProtocolClause msg){
  if ( testmessageFromAToBtextText3AencryptKeyABnonceNBBagentBtextText2A (msg) )
  {messageFromAToBtextText3AencryptKeyABnonceNBBagentBtextText2AQuery = True ;

     }
     else  { skip  ;

       }
   /* End of test method */

  }
   /* End of receive method */

  }
  /* End of class */



 // The protocol: ISO-SYM-Two-Pass-unilateral Authentication
 {
 Agent zombieAgent =  new cog ZombieAgent(AgentName("TestZombie") )  ;

   Network network  ;

   network =  new cog Network(zombieAgent)  ;

   Agent objectA ;

   objectA = new cog Aclass (AgentVar( "A" ) , network) ;

   network!register(AgentVar( "A" ), objectA)  ;

   Agent objectB ;

   objectB = new cog Bclass (AgentVar( "B" ) , network) ;

   network!register(AgentVar( "B" ), objectB)  ;

   }
 // end of file
```

ISO One two-pass mutual

```
module TestProject;

import * from ABSProtocols;
import * from ExecutionEnvironment;

 class  Aclass (AgentTerm name, Network network) implements Agent{
  PayloadElement textText2A =  UndefinedElem   ;

    PayloadElement newnonceNAA =  UndefinedElem   ;

    PayloadElement agentB =  UndefinedElem   ;

    PayloadElement textText1A =  UndefinedElem   ;

    PayloadElement keyKeyAB =  UndefinedElem   ;

    PayloadElement encryptKeyABnonceNAAagentBtextText1A =  UndefinedElem   ;

    ProtocolClause messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A =
      UndefinedClause  ;

    PayloadElement textText4A =  UndefinedElem   ;

    PayloadElement nonceNBB =  UndefinedElem   ;

    PayloadElement agentA =  UndefinedElem   ;
```

```
    PayloadElement textText3B =  UndefinedElem    ;

    PayloadElement encryptKeyABnonceNBBagentAtextText3B =
      UndefinedElem    ;

    ProtocolClause
      messageFromAToBtextText4AencryptKeyABnonceNBBagentAtextText3B =
         UndefinedClause   ;

      Unit run(){ await(network != null) ;

     textText2A =  TextOrigin("Text2", AgentVar( "A" ))   ;

   newnonceNAA =  New(Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ) ) ;

   agentB =  Agent(AgentVar( "B" )) ;

   textText1A =  TextOrigin("Text1", AgentVar( "A" ))   ;

   keyKeyAB =  Key(KeySymmetric( AgentVar( "A" ),AgentVar( "B" ))) ;

   encryptKeyABnonceNAAagentBtextText1A =
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
  list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
   Agent(AgentVar( "B" )),  TextOrigin("Text1", AgentVar( "A" )) ] ) ;

   messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A =
Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text2", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
      Agent(AgentVar( "B" )),  TextOrigin("Text1", AgentVar( "A" )) ] )]) ;

   network!send( Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text2", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
      Agent(AgentVar( "B" )),  TextOrigin("Text1", AgentVar( "A" )) ] )]) )  ;

   textText4A =  TextOrigin("Text4", AgentVar( "A" ))   ;

   nonceNBB = Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ) ;

   agentA =  Agent(AgentVar( "A" )) ;

   textText3B =  TextOrigin("Text3", AgentVar( "B" ))   ;

   encryptKeyABnonceNBBagentAtextText3B =
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
      Agent(AgentVar( "A" )),  TextOrigin("Text3", AgentVar( "B" )) ] ) ;

   messageFromAToBtextText4AencryptKeyABnonceNBBagentAtextText3B =
Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text4", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
      Agent(AgentVar( "A" )),  TextOrigin("Text3", AgentVar( "B" )) ] )]) ;

   network!send( Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text4", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
    list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
          Agent(AgentVar( "A" )),  TextOrigin("Text3", AgentVar( "B" )) ] )]) )  ;

    }
   /* End of run-method */


    Unit receive(ProtocolClause msg){skip  ;
```

```
    }
   /* End of receive method */

  }
  /* End of class */

  def Bool
   testmessageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A
                    (ProtocolClause msg)
= case msg {Message( AgentVar( "A" ),    AgentVar( "B" ),

       Cons (  TextOrigin("Text2", AgentVar( "A" )) ,
       Cons (  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
       Cons ( Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
       Cons (  Agent(AgentVar( "B" )),
       Cons (  TextOrigin("Text1", AgentVar( "A" )) ,  Nil
 )  )  )  ) ,  Nil
 )  )  )     => True ;

     _ => False ;

     }  ;


  def Bool
    testmessageFromAToBtextText4AencryptKeyABnonceNBBagentAtextText3B
                  (ProtocolClause msg)
= case msg {Message( AgentVar( "A" ),    AgentVar( "B" ),

       Cons (  TextOrigin("Text4", AgentVar( "A" )) ,
       Cons (  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
       Cons ( Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
       Cons (  Agent(AgentVar( "A" )),
       Cons (  TextOrigin("Text3", AgentVar( "B" )) ,  Nil
 )  )  )  ) ,  Nil
 )  )  )     => True ;

     _ => False ;

     }  ;



class  Bclass (AgentTerm name, Network network) implements Agent{
Bool
  messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1AQuery = False  ;

Bool
 messageFromAToBtextText4AencryptKeyABnonceNBBagentAtextText3BQuery = False  ;

    ProtocolClause
     messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A =
      UndefinedClause   ;

    PayloadElement agentA =  UndefinedElem   ;

    PayloadElement textText2A =  UndefinedElem   ;

    PayloadElement keyKeyAB =  UndefinedElem   ;

    PayloadElement
      decryptKeyABencryptKeyABnonceNAAagentBtextText1A =
        UndefinedElem   ;

    PayloadElement nonceNAA =  UndefinedElem   ;

    PayloadElement agentB =  UndefinedElem   ;

    PayloadElement textText1A =  UndefinedElem   ;

    ProtocolClause
```

```
      messageFromAToBtextText4AencryptKeyABnonceNBBagentAtextText3B =
        UndefinedClause    ;

     PayloadElement textText4A =  UndefinedElem    ;

     PayloadElement decryptKeyABencryptKeyABnonceNBBagentAtextText3B
         =  UndefinedElem    ;

     PayloadElement newnonceNBB =  UndefinedElem    ;

     PayloadElement textText3B =  UndefinedElem    ;

       Unit run(){ await(network != null) ;

      messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A =
Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text2", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),  AgentVar( "B" )),
   list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
         Agent(AgentVar( "B" )),  TextOrigin("Text1", AgentVar( "A" )) ] )]) ;

   await( messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1AQuery )  ;

   agentA =  Agent(AgentVar( "A" )) ;

   textText2A =  TextOrigin("Text2", AgentVar( "A" ))  ;

   keyKeyAB =  Key(KeySymmetric( AgentVar( "A" ),AgentVar( "B" ))) ;

   decryptKeyABencryptKeyABnonceNAAagentBtextText1A =
  Decrypt(  KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
  list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),
        Agent(AgentVar( "B" )),  TextOrigin("Text1", AgentVar( "A" )) ] ) ] ) ;

   nonceNAA = Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ) ;

   agentB =  Agent(AgentVar( "B" )) ;

   textText1A =  TextOrigin("Text1", AgentVar( "A" ))  ;

   messageFromAToBtextText4AencryptKeyABnonceNBBagentAtextText3B =
    Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text4", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
   list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
         Agent(AgentVar( "A" )),  TextOrigin("Text3", AgentVar( "B" )) ] )]) ;

   await( messageFromAToBtextText4AencryptKeyABnonceNBBagentAtextText3BQuery )  ;

   textText4A =  TextOrigin("Text4", AgentVar( "A" ))  ;

   decryptKeyABencryptKeyABnonceNBBagentAtextText3B =
  Decrypt(  KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
    list[ Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ),
          Agent(AgentVar( "A" )),  TextOrigin("Text3", AgentVar( "B" )) ] ) ] ) ;

   newnonceNBB =  New(Nonce( NonceVar ( "NB" ), AgentVar( "B" ) ) ) ;

   textText3B =  TextOrigin("Text3", AgentVar( "B" ))  ;

     }
   /* End of run-method */


    Unit receive(ProtocolClause msg){
  if ( testmessageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1A (msg) )
  {messageFromAToBtextText2AencryptKeyABnonceNAAagentBtextText1AQuery = True ;

     }
     else  {
```

```
  if ( testmessageFromAToBtextText4AencryptKeyABnonceNBBagentAtextText3B (msg) )
  {messageFromAToBtextText4AencryptKeyABnonceNBBagentAtextText3BQuery = True ;

     }
     else  { skip  ;

       }
   /* End of test method */

     }
   /* End of test method */

  }
   /* End of receive method */

  }
  /* End of class */



 // The protocol: ISO-SYM-Two-Pass mutual  Authentication
 {
 Agent zombieAgent =  new cog ZombieAgent(AgentName("TestZombie") )  ;

   Network network  ;

   network =  new cog Network(zombieAgent)  ;

   Agent objectA ;

   objectA = new cog Aclass (AgentVar( "A" ) , network) ;

   network!register(AgentVar( "A" ), objectA)  ;

   Agent objectB ;

   objectB = new cog Bclass (AgentVar( "B" ) , network) ;

   network!register(AgentVar( "B" ), objectB)  ;

   }
 // end of file
```

ISO-SYM-threepass-mutual.abs

```
 module TestProject;

import * from ABSProtocols;
import * from ExecutionEnvironment;

  def Bool testmessageFromBToAnonceRBBtextText1B (ProtocolClause msg)
 = case msg {Message( AgentVar( "B" ),    AgentVar( "A" ),

       Cons ( Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ),
       Cons (  TextOrigin("Text1", AgentVar( "B" )) ,  Nil
  )  )  )     => True ;

     _ => False ;

     }  ;



 class  Aclass (AgentTerm name, Network network) implements Agent{
 Bool messageFromBToAnonceRBBtextText1BQuery = False  ;

    ProtocolClause messageFromBToAnonceRBBtextText1B =  UndefinedClause    ;

    PayloadElement agentB =  UndefinedElem    ;

    PayloadElement nonceRBB =  UndefinedElem    ;
```

```
    PayloadElement textText1B = UndefinedElem ;

    PayloadElement textText3A = UndefinedElem ;

    PayloadElement newnonceRAA = UndefinedElem ;

    PayloadElement textText2A = UndefinedElem ;

    PayloadElement keyKeyAB = UndefinedElem ;

    PayloadElement encryptKeyABnonceRAAnonceRBBagentBtextText2A = UndefinedElem ;

    ProtocolClause
  messageFromAToBtextText3AencryptKeyABnonceRAAnonceRBBagentBtextText2A =
          UndefinedClause ;

    PayloadElement textText5A = UndefinedElem ;

    PayloadElement agentA = UndefinedElem ;

    PayloadElement textText4A = UndefinedElem ;

    PayloadElement
     encryptKeyABnonceRBBnonceRAAagentAtextText4A = UndefinedElem ;

    ProtocolClause
 messageFromAToBtextText5AencryptKeyABnonceRBBnonceRAAagentAtextText4A =
          UndefinedClause ;

      Unit run(){ await(network != null) ;

     messageFromBToAnonceRBBtextText1B = Message(AgentVar( "B" ),AgentVar( "A" ),
  list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ), TextOrigin("Text1", AgentVar( "B" )) ]) ;

   await( messageFromBToAnonceRBBtextText1BQuery ) ;

   agentB = Agent(AgentVar( "B" )) ;

   nonceRBB = Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ) ;

   textText1B = TextOrigin("Text1", AgentVar( "B" )) ;

   textText3A = TextOrigin("Text3", AgentVar( "A" )) ;

   newnonceRAA = New(Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ) ) ;

   textText2A = TextOrigin("Text2", AgentVar( "A" )) ;

   keyKeyAB = Key(KeySymmetric( AgentVar( "A" ),AgentVar( "B" ))) ;

   encryptKeyABnonceRAAnonceRBBagentBtextText2A =
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
    list[ Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ), Nonce( NonceVar ( "RB" ),
        AgentVar( "B" ) ), Agent(AgentVar( "B" )), TextOrigin("Text2", AgentVar( "A" )) ] ) ;

   messageFromAToBtextText3AencryptKeyABnonceRAAnonceRBBagentBtextText2A =
Message(AgentVar( "A" ),AgentVar( "B" ),
  list[ TextOrigin("Text3", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
 list[ Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ), Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ),
       Agent(AgentVar( "B" )), TextOrigin("Text2", AgentVar( "A" )) ] )]) ;

   network!send( Message(AgentVar( "A" ),AgentVar( "B" ),
  list[ TextOrigin("Text3", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
 list[ Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ), Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ),
       Agent(AgentVar( "B" )), TextOrigin("Text2", AgentVar( "A" )) ] )]) ) ;

   textText5A = TextOrigin("Text5", AgentVar( "A" )) ;

   agentA = Agent(AgentVar( "A" )) ;
```

```
   textText4A =  TextOrigin("Text4", AgentVar( "A" ))  ;

   encryptKeyABnonceRBBnonceRAAagentAtextText4A =
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
   list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ), Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),
         Agent(AgentVar( "A" )),  TextOrigin("Text4", AgentVar( "A" )) ] ) ;

   messageFromAToBtextText5AencryptKeyABnonceRBBnonceRAAagentAtextText4A =
 Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text5", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
    list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ), Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),
          Agent(AgentVar( "A" )),  TextOrigin("Text4", AgentVar( "A" )) ] )]) ;

   network!send( Message(AgentVar( "A" ),AgentVar( "B" ),
  list[  TextOrigin("Text5", AgentVar( "A" )) ,
  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
    list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ), Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),
          Agent(AgentVar( "A" )),  TextOrigin("Text4", AgentVar( "A" )) ] )]) )  ;

     }
   /* End of run-method */


    Unit receive(ProtocolClause msg){
  if ( testmessageFromBToAnonceRBBtextText1B (msg) )
  {messageFromBToAnonceRBBtextText1BQuery = True ;

     }
     else  { skip  ;

       }
   /* End of test method */

  }
   /* End of receive method */

  }
  /* End of class */

  def Bool
testmessageFromAToBtextText3AencryptKeyABnonceRAAnonceRBBagentBtextText2A (ProtocolClause msg)
 = case msg {Message( AgentVar( "A" ),   AgentVar( "B" ),

       Cons (  TextOrigin("Text3", AgentVar( "A" )) ,
       Cons (  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
       Cons ( Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),
       Cons ( Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ),
       Cons (  Agent(AgentVar( "B" )),
       Cons (  TextOrigin("Text2", AgentVar( "A" )) ,  Nil
  ) ) ) ) ) ) , Nil
  ) ) )     => True ;

     _ => False ;

     }  ;


  def Bool
testmessageFromAToBtextText5AencryptKeyABnonceRBBnonceRAAagentAtextText4A (ProtocolClause msg)
 = case msg {Message( AgentVar( "A" ),   AgentVar( "B" ),

       Cons (  TextOrigin("Text5", AgentVar( "A" )) ,
       Cons (  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
       Cons ( Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ),
       Cons ( Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),
       Cons (  Agent(AgentVar( "A" )),
       Cons (  TextOrigin("Text4", AgentVar( "A" )) ,  Nil
  ) ) ) ) ) ) , Nil
  ) ) )     => True ;
```

```
    _ => False ;

    } ;


class Bclass (AgentTerm name, Network network) implements Agent{
Bool messageFromAToBtextText3AencryptKeyABnonceRAAnonceRBBagentBtextText2AQuery = False ;

  Bool messageFromAToBtextText5AencryptKeyABnonceRBBnonceRAAagentAtextText4AQuery = False ;

   PayloadElement newnonceRBB = UndefinedElem ;

   PayloadElement textText1B = UndefinedElem ;

   PayloadElement agentA = UndefinedElem ;

   ProtocolClause messageFromBToAnonceRBBtextText1B = UndefinedClause ;

   ProtocolClause
 messageFromAToBtextText3AencryptKeyABnonceRAAnonceRBBagentBtextText2A =
   UndefinedClause ;

   PayloadElement textText3A = UndefinedElem ;

   PayloadElement keyKeyAB = UndefinedElem ;

   PayloadElement
 decryptKeyABencryptKeyABnonceRAAnonceRBBagentBtextText2A = UndefinedElem ;

   PayloadElement nonceRAA = UndefinedElem ;

   PayloadElement agentB = UndefinedElem ;

   PayloadElement textText2A = UndefinedElem ;

   ProtocolClause
 messageFromAToBtextText5AencryptKeyABnonceRBBnonceRAAagentAtextText4A = UndefinedClause ;

   PayloadElement textText5A = UndefinedElem ;

   PayloadElement decryptKeyABencryptKeyABnonceRBBnonceRAAagentAtextText4A = UndefinedElem ;

   PayloadElement textText4A = UndefinedElem ;

     Unit run(){ await(network != null) ;

    newnonceRBB = New(Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ) ) ;

  textText1B = TextOrigin("Text1", AgentVar( "B" )) ;

  agentA = Agent(AgentVar( "A" )) ;

  messageFromBToAnonceRBBtextText1B = Message(AgentVar( "B" ),AgentVar( "A" ),
 list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ), TextOrigin("Text1", AgentVar( "B" )) ]) ;

  network!send( Message(AgentVar( "B" ),AgentVar( "A" ),
 list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ), TextOrigin("Text1", AgentVar( "B" )) ]) ) ;

  messageFromAToBtextText3AencryptKeyABnonceRAAnonceRBBagentBtextText2A =
 Message(AgentVar( "A" ),AgentVar( "B" ),
 list[ TextOrigin("Text3", AgentVar( "A" )) ,
 Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
 list[ Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ), Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ),
Agent(AgentVar( "B" )), TextOrigin("Text2", AgentVar( "A" )) ] )]) ;

  await( messageFromAToBtextText3AencryptKeyABnonceRAAnonceRBBagentBtextText2AQuery ) ;

  textText3A = TextOrigin("Text3", AgentVar( "A" )) ;

  keyKeyAB = Key(KeySymmetric( AgentVar( "A" ),AgentVar( "B" ))) ;
```

```
  decryptKeyABencryptKeyABnonceRAAnonceRBBagentBtextText2A =
 Decrypt(  KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
 Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
   list[ Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ), Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ),
   Agent(AgentVar( "B" )),  TextOrigin("Text2", AgentVar( "A" )) ] ) ] ) ;

  nonceRAA = Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ) ;

  agentB =  Agent(AgentVar( "B" )) ;

  textText2A =  TextOrigin("Text2", AgentVar( "A" ))  ;

  messageFromAToBtextText5AencryptKeyABnonceRBBnonceRAAagentAtextText4A =
  Message(AgentVar( "A" ),AgentVar( "B" ),
 list[  TextOrigin("Text5", AgentVar( "A" )) ,
 Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ), Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),
 Agent(AgentVar( "A" )),  TextOrigin("Text4", AgentVar( "A" )) ] )]) ;

  await( messageFromAToBtextText5AencryptKeyABnonceRBBnonceRAAagentAtextText4AQuery )  ;

  textText5A =  TextOrigin("Text5", AgentVar( "A" ))  ;

  decryptKeyABencryptKeyABnonceRBBnonceRAAagentAtextText4A =
 Decrypt(  KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
 Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
   list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ), Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),
         Agent(AgentVar( "A" )),  TextOrigin("Text4", AgentVar( "A" )) ] ) ] ) ;

  textText4A =  TextOrigin("Text4", AgentVar( "A" ))  ;

    }
  /* End of run-method */


   Unit receive(ProtocolClause msg){
 if ( testmessageFromAToBtextText3AencryptKeyABnonceRAAnonceRBBagentBtextText2A (msg) )
 {messageFromAToBtextText3AencryptKeyABnonceRAAnonceRBBagentBtextText2AQuery = True ;

    }
    else  {
 if ( testmessageFromAToBtextText5AencryptKeyABnonceRBBnonceRAAagentAtextText4A (msg) )
 {messageFromAToBtextText5AencryptKeyABnonceRBBnonceRAAagentAtextText4AQuery = True ;

    }
    else  { skip  ;

      }
  /* End of test method */

    }
  /* End of test method */

 }
  /* End of receive method */

 }
 /* End of class */


// The protocol: ISO-SYM-Two-Pass-unilateral Authentication
{
Agent zombieAgent =  new cog ZombieAgent(AgentName("TestZombie") )  ;

  Network network  ;

  network =  new cog Network(zombieAgent)  ;

  Agent objectA ;
```

```
  objectA = new cog Aclass (AgentVar( "A" ) , network) ;

  network!register(AgentVar( "A" ), objectA)  ;

  Agent objectB ;

  objectB = new cog Bclass (AgentVar( "B" ) , network) ;

  network!register(AgentVar( "B" ), objectB)  ;

  }
// end of file
```

Andrew Secure

```
 def Bool testmessageFromYToXencryptKeyXYnonceincrementNXXnonceNYY (ProtocolClause msg)
= case msg {Message( AgentVar( "Y" ),   AgentVar( "X" ),

      Cons (  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
      Cons ( Nonce( NonceVar (NX + 1 ), AgentVar( "X" ) ),
      Cons ( Nonce( NonceVar ( "NY" ), AgentVar( "Y" ) ),  Nil
 ) ) ) ,  Nil
 ) )     => True ;

    _ => False ;

    }  ;


 def Bool testmessageFromYToXencryptKeyXYkeyKeyXYKVnonceNY2Y (ProtocolClause msg)
= case msg {Message( AgentVar( "Y" ),   AgentVar( "X" ),

      Cons (  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
      Cons (  Key( KeySymmetricFresh( AgentVar( "X" ),AgentVar( "Y" ),KeyVar( "KV"  ) )),
      Cons ( Nonce( NonceVar ( "NY2" ), AgentVar( "Y" ) ),  Nil
 ) ) ) ,  Nil
 ) )     => True ;

    _ => False ;

    }  ;



class  Xclass (AgentTerm name, Network network) implements Agent{
Bool messageFromYToXencryptKeyXYnonceincrementNXXnonceNYYQuery = False  ;

  Bool messageFromYToXencryptKeyXYkeyKeyXYKVnonceNY2YQuery = False  ;

   PayloadElement agentX =  UndefinedElem   ;

   PayloadElement newnonceNXX =  UndefinedElem   ;

   PayloadElement keyKeyXY =  UndefinedElem   ;

   PayloadElement encryptKeyXYnonceNXX =  UndefinedElem   ;

   PayloadElement agentY =  UndefinedElem   ;

   ProtocolClause messageFromXToYagentXencryptKeyXYnonceNXX =  UndefinedClause  ;

   ProtocolClause messageFromYToXencryptKeyXYnonceincrementNXXnonceNYY =  UndefinedClause   ;

   PayloadElement decryptKeyXYencryptKeyXYnonceincrementNXXnonceNYY =  UndefinedElem   ;

   PayloadElement nonceincrementNXX =  UndefinedElem   ;

   PayloadElement nonceNYY =  UndefinedElem   ;

   PayloadElement nonceincrementNYY =  UndefinedElem   ;
```

```
    PayloadElement encryptKeyXYnonceincrementNYY = UndefinedElem ;

    ProtocolClause messageFromXToYencryptKeyXYnonceincrementNYY = UndefinedClause ;

    ProtocolClause messageFromYToXencryptKeyXYkeyKeyXYKVnonceNY2Y = UndefinedClause ;

    PayloadElement decryptKeyXYencryptKeyXYkeyKeyXYKVnonceNY2Y = UndefinedElem ;

    PayloadElement newkeyKeyXYKV = UndefinedElem ;

    PayloadElement nonceNY2Y = UndefinedElem ;

      Unit run(){ await(network != null) ;

     agentX = Agent(AgentVar( "X" )) ;

  newnonceNXX = New(Nonce( NonceVar ( "NX" ), AgentVar( "X" ) ) ) ;

  keyKeyXY = Key(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" ))) ;

  encryptKeyXYnonceNXX =
 Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
     list[ Nonce( NonceVar ( "NX" ), AgentVar( "X" ) )] ) ;

  agentY = Agent(AgentVar( "Y" )) ;

  messageFromXToYagentXencryptKeyXYnonceNXX = Message(AgentVar( "X" ),AgentVar( "Y" ),
 list[ Agent(AgentVar( "X" )),
 Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
    list[ Nonce( NonceVar ( "NX" ), AgentVar( "X" ) )] )]) ;

  network!send( Message(AgentVar( "X" ),AgentVar( "Y" ),
 list[ Agent(AgentVar( "X" )),
 Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
    list[ Nonce( NonceVar ( "NX" ), AgentVar( "X" ) )] )]) ) ;

  messageFromYToXencryptKeyXYnonceincrementNXXnonceNYY =
Message(AgentVar( "Y" ),AgentVar( "X" ),
 list[
 Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
list[ Nonce( NonceVar (NX + 1 ), AgentVar( "X" ) ),
    Nonce( NonceVar ( "NY" ), AgentVar( "Y" ) )] )]) ;

  await( messageFromYToXencryptKeyXYnonceincrementNXXnonceNYYQuery ) ;

  decryptKeyXYencryptKeyXYnonceincrementNXXnonceNYY =
 Decrypt( KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )), list[
 Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
  list[ Nonce( NonceVar (NX + 1 ), AgentVar( "X" ) ),
    Nonce( NonceVar ( "NY" ), AgentVar( "Y" ) )] ) ] ) ;

  nonceincrementNXX = Nonce( NonceVar (NX + 1 ), AgentVar( "X" ) ) ;

  nonceNYY = Nonce( NonceVar ( "NY" ), AgentVar( "Y" ) ) ;

  nonceincrementNYY = Nonce( NonceVar (NY + 1 ), AgentVar( "Y" ) ) ;

  encryptKeyXYnonceincrementNYY =
 Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
    list[ Nonce( NonceVar (NY + 1 ), AgentVar( "Y" ) )] ) ;

  messageFromXToYencryptKeyXYnonceincrementNYY =
Message(AgentVar( "X" ),AgentVar( "Y" ),
 list[
 Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
    list[ Nonce( NonceVar (NY + 1 ), AgentVar( "Y" ) )] )]) ;

  network!send( Message(AgentVar( "X" ),AgentVar( "Y" ),
 list[
 Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
   list[ Nonce( NonceVar (NY + 1 ), AgentVar( "Y" ) )] )]) ) ;
```

```
  messageFromYToXencryptKeyXYkeyKeyXYKVnonceNY2Y =
    Message(AgentVar( "Y" ),AgentVar( "X" ),
  list[
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
   list[ Key( KeySymmetricFresh( AgentVar( "X" ),AgentVar( "Y" ),KeyVar( "KV") )),
           Nonce( NonceVar ( "NY2" ), AgentVar( "Y" ) )] )]) ;

   await( messageFromYToXencryptKeyXYkeyKeyXYKVnonceNY2YQuery )  ;

   decryptKeyXYencryptKeyXYkeyKeyXYKVnonceNY2Y =
  Decrypt(  KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )), list[
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
    list[  Key( KeySymmetricFresh( AgentVar( "X" ),AgentVar( "Y" ),KeyVar( "KV"  ) )),
    Nonce( NonceVar ( "NY2" ), AgentVar( "Y" ) )] ) ] ) ;

   newkeyKeyXYKV =
New(Key( KeySymmetricFresh( AgentVar( "X" ), AgentVar( "Y" ),KeyVar( "KV"  ) ))) ;

   nonceNY2Y = Nonce( NonceVar ( "NY2" ), AgentVar( "Y" ) ) ;

     }
   /* End of run-method */


    Unit receive(ProtocolClause msg){
  if ( testmessageFromYToXencryptKeyXYnonceincrementNXXnonceNYY (msg) )
  {messageFromYToXencryptKeyXYnonceincrementNXXnonceNYYQuery = True ;

     }
     else  {
  if ( testmessageFromYToXencryptKeyXYkeyKeyXYKVnonceNY2Y (msg) )
  {messageFromYToXencryptKeyXYkeyKeyXYKVnonceNY2YQuery = True ;

     }
     else  { skip  ;

       }
   /* End of test method */

     }
   /* End of test method */

  }
   /* End of receive method */

  }
  /* End of class */

  def Bool testmessageFromXToYagentXencryptKeyXYnonceNXX (ProtocolClause msg)
= case msg {Message( AgentVar( "X" ),   AgentVar( "Y" ),

       Cons (  Agent(AgentVar( "X" )),
       Cons (  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
       Cons ( Nonce( NonceVar ( "NX" ), AgentVar( "X" ) ),  Nil
  ) ) ,  Nil
  ) ) )     => True ;

     _ => False ;

     }  ;


  def Bool testmessageFromXToYencryptKeyXYnonceincrementNYY (ProtocolClause msg)
= case msg {Message( AgentVar( "X" ),   AgentVar( "Y" ),

       Cons (  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
       Cons ( Nonce( NonceVar (NY + 1 ), AgentVar( "Y" ) ),  Nil
  ) ) ,  Nil
  ) )     => True ;

     _ => False ;
```

```
    } ;


class Yclass (AgentTerm name, Network network) implements Agent{
Bool messageFromXToYagentXencryptKeyXYnonceNXXQuery = False ;

  Bool messageFromXToYencryptKeyXYnonceincrementNYYQuery = False ;

   ProtocolClause messageFromXToYagentXencryptKeyXYnonceNXX = UndefinedClause ;

   PayloadElement agentX = UndefinedElem ;

   PayloadElement keyKeyXY = UndefinedElem ;

   PayloadElement decryptKeyXYencryptKeyXYnonceNXX = UndefinedElem ;

   PayloadElement nonceNXX = UndefinedElem ;

   PayloadElement nonceincrementNXX = UndefinedElem ;

   PayloadElement newnonceNYY = UndefinedElem ;

   PayloadElement encryptKeyXYnonceincrementNXXnonceNYY = UndefinedElem ;

   ProtocolClause messageFromYToXencryptKeyXYnonceincrementNXXnonceNYY = UndefinedClause ;

   ProtocolClause messageFromXToYencryptKeyXYnonceincrementNYY = UndefinedClause ;

   PayloadElement decryptKeyXYencryptKeyXYnonceincrementNYY = UndefinedElem ;

   PayloadElement nonceincrementNYY = UndefinedElem ;

   PayloadElement newkeyKeyXYKV = UndefinedElem ;

   PayloadElement newkeyKeyXYKV = UndefinedElem ;

   PayloadElement newnonceNY2Y = UndefinedElem ;

   PayloadElement encryptKeyXYkeyKeyXYKVnonceNY2Y = UndefinedElem ;

   ProtocolClause messageFromYToXencryptKeyXYkeyKeyXYKVnonceNY2Y = UndefinedClause ;

     Unit run(){ await(network != null) ;

    messageFromXToYagentXencryptKeyXYnonceNXX = Message(AgentVar( "X" ),AgentVar( "Y" ),
  list[ Agent(AgentVar( "X" )),
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
    list[ Nonce( NonceVar ( "NX" ), AgentVar( "X" ) )] )]) ;

   await( messageFromXToYagentXencryptKeyXYnonceNXXQuery ) ;

   agentX = Agent(AgentVar( "X" )) ;

   keyKeyXY = Key(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" ))) ;

   decryptKeyXYencryptKeyXYnonceNXX =
  Decrypt( KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )), list[
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
     list[ Nonce( NonceVar ( "NX" ), AgentVar( "X" ) )] ) ] ) ;

   nonceNXX = Nonce( NonceVar ( "NX" ), AgentVar( "X" ) ) ;

   nonceincrementNXX = Nonce( NonceVar (NX + 1 ), AgentVar( "X" ) ) ;

   newnonceNYY = New(Nonce( NonceVar ( "NY" ), AgentVar( "Y" ) ) ) ;

   encryptKeyXYnonceincrementNXXnonceNYY =
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
list[ Nonce( NonceVar (NX + 1 ), AgentVar( "X" ) ),
```

```
      Nonce( NonceVar ( "NY" ), AgentVar( "Y" ) )] ) ;

   messageFromYToXencryptKeyXYnonceincrementNXXnonceNYY
   = Message(AgentVar( "Y" ),AgentVar( "X" ),
  list[
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
list[ Nonce( NonceVar (NX + 1 ), AgentVar( "X" ) ),
               Nonce( NonceVar ( "NY" ), AgentVar( "Y" ) )] )]) ;

   network!send( Message(AgentVar( "Y" ),AgentVar( "X" ),
  list[
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
list[ Nonce( NonceVar (NX + 1 ), AgentVar( "X" ) ),
       Nonce( NonceVar ( "NY" ), AgentVar( "Y" ) )] )]) )  ;

   messageFromXToYencryptKeyXYnonceincrementNYY =
     Message(AgentVar( "X" ),AgentVar( "Y" ),
  list[
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
list[ Nonce( NonceVar (NY + 1 ), AgentVar( "Y" ) )] )]) ;

   await( messageFromXToYencryptKeyXYnonceincrementNYYQuery )  ;

   decryptKeyXYencryptKeyXYnonceincrementNYY =
  Decrypt(  KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )), list[
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
       list[ Nonce( NonceVar (NY + 1 ), AgentVar( "Y" ) )] ) ] ) ;

   nonceincrementNYY = Nonce( NonceVar (NY + 1 ), AgentVar( "Y" ) ) ;

   newkeyKeyXYKV =
New(Key( KeySymmetricFresh( AgentVar( "X" ),AgentVar( "Y" ),KeyVar( "KV"  ) ))) ;

   newkeyKeyXYKV =
New(Key( KeySymmetricFresh( AgentVar( "X" ),AgentVar( "Y" ),KeyVar( "KV"  ) ))) ;

   newnonceNY2Y =  New(Nonce( NonceVar ( "NY2" ), AgentVar( "Y" ) ) ) ;

   encryptKeyXYkeyKeyXYKVnonceNY2Y =
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
   list[  Key( KeySymmetricFresh( AgentVar( "X" ),AgentVar( "Y" ),KeyVar( "KV"  ) )),
      Nonce( NonceVar ( "NY2" ), AgentVar( "Y" ) )] ) ;

   messageFromYToXencryptKeyXYkeyKeyXYKVnonceNY2Y =
Message(AgentVar( "Y" ),AgentVar( "X" ),
  list[
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
    list[  Key( KeySymmetricFresh( AgentVar( "X" ),AgentVar( "Y" ),KeyVar( "KV"  ) )),
           Nonce( NonceVar ( "NY2" ), AgentVar( "Y" ) )] )]) ;

   network!send( Message(AgentVar( "Y" ),AgentVar( "X" ),
  list[
  Encrypt(KeySymmetric( AgentVar( "X" ),AgentVar( "Y" )),
    list[  Key( KeySymmetricFresh( AgentVar( "X" ),AgentVar( "Y" ),KeyVar( "KV"  ) )),
           Nonce( NonceVar ( "NY2" ), AgentVar( "Y" ) )] )]) )  ;

     }
   /* End of run-method */


    Unit receive(ProtocolClause msg){
  if ( testmessageFromXToYagentXencryptKeyXYnonceNXX (msg) )
  {messageFromXToYagentXencryptKeyXYnonceNXXQuery = True ;

     }
     else  {
  if ( testmessageFromXToYencryptKeyXYnonceincrementNYY (msg) )
  {messageFromXToYencryptKeyXYnonceincrementNYYQuery = True ;

     }
     else  { skip  ;
```

```
      }
  /* End of test method */

    }
  /* End of test method */

 }
  /* End of receive method */

 }
 /* End of class */



// The protocol: Andrew Secure
{
Agent zombieAgent =  new cog ZombieAgent(AgentName("TestZombie") )  ;

  Network network  ;

  network =  new cog Network(zombieAgent)  ;

  Agent objectX ;

  objectX = new cog Xclass (AgentVar( "X" ) , network) ;

  network!register(AgentVar( "X" ), objectX)  ;

  Agent objectY ;

  objectY = new cog Yclass (AgentVar( "Y" ) , network) ;

  network!register(AgentVar( "Y" ), objectY)  ;

  }
// end of file
```

Non reversible functions

```
 def Bool testmessageFromBToAagentBnonceRBB (ProtocolClause msg)
= case msg {Message( AgentVar( "B" ),   AgentVar( "A" ),

      Cons (  Agent(AgentVar( "B" )),
      Cons ( Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ),  Nil
 )  )  )     => True ;

    _ => False ;

    }  ;


 def Bool
 testmessageFromBToAagentBencryptKeyCompositetextsymmetrickeyABAhashnonceNAA (ProtocolClause msg)
= case msg {Message( AgentVar( "B" ),   AgentVar( "A" ),

      Cons (  Agent(AgentVar( "B" )),
      Cons (  Encrypt(KeyComposite (
      Cons (  TextOrigin("symmetric_key_AB", AgentVar( "A" )) ,  Nil
 ) ),
      Cons (  Hash(
      Cons ( Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ),  Nil
 )  ) ,  Nil
 )  ) ,  Nil
 )  )  )     => True ;

    _ => False ;

    }  ;
```

```
class  Aclass (AgentTerm name, Network network) implements Agent{
Bool messageFromBToAagentBnonceRBBQuery = False  ;

  Bool messageFromBToAagentBencryptKeyCompositetextsymmetrickeyABAhashnonceNAAQuery = False  ;

   ProtocolClause messageFromBToAagentBnonceRBB =  UndefinedClause   ;

   PayloadElement agentB =  UndefinedElem   ;

   PayloadElement nonceRBB =  UndefinedElem   ;

   PayloadElement textsymmetric key ABA =  UndefinedElem   ;

   PayloadElement textsymmetrickeyABA =  UndefinedElem   ;

   PayloadElement keyKeyCompositetextsymmetrickeyABA =  UndefinedElem   ;

   PayloadElement agentA =  UndefinedElem   ;

   PayloadElement hashnonceRBB =  UndefinedElem   ;

   PayloadElement newnonceRAA =  UndefinedElem   ;

   PayloadElement keyKeyAB =  UndefinedElem   ;

   PayloadElement
     encryptKeyABhashnonceRBBnonceRAAagentAkeyKeyCompositetextsymmetrickeyABA
         =  UndefinedElem   ;

   ProtocolClause
messageFromAToBagentAencryptKeyABhashnonceRBBnonceRAAagentAkeyKeyCompositetextsymmetrickeyABA
       =  UndefinedClause  ;

   ProtocolClause
messageFromBToAagentBencryptKeyCompositetextsymmetrickeyABAhashnonceNAA
       =  UndefinedClause   ;

   PayloadElement
decryptKeyCompositetextsymmetrickeyABAencryptKeyCompositetextsymmetrickeyABAhashnonceNAA
       =  UndefinedElem   ;

   PayloadElement hashnonceNAA =  UndefinedElem   ;

     Unit run(){ await(network != null) ;

    messageFromBToAagentBnonceRBB = Message(AgentVar( "B" ),AgentVar( "A" ),
 list[  Agent(AgentVar( "B" )), Nonce( NonceVar ( "RB" ), AgentVar( "B" ) )]) ;

  await( messageFromBToAagentBnonceRBBQuery )  ;

  agentB =  Agent(AgentVar( "B" )) ;

  nonceRBB = Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ) ;

  textsymmetric key ABA = New( TextOrigin("symmetric_key_AB",AgentVar( "A" )) )  ;

  textsymmetrickeyABA =  TextOrigin("symmetric_key_AB", AgentVar( "A" ))  ;

  keyKeyCompositetextsymmetrickeyABA =
    Key( KeyComposite(list[   TextOrigin("symmetric_key_AB", AgentVar( "A" )) ] ) ) ;

  agentA =  Agent(AgentVar( "A" )) ;

  hashnonceRBB =
Hash( list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) )] ) ;

  newnonceRAA =  New(Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ) ) ;

  keyKeyAB =  Key(KeySymmetric( AgentVar( "A" ),AgentVar( "B" ))) ;
```

```
  encryptKeyABhashnonceRBBnonceRAAagentAkeyKeyCompositetextsymmetrickeyABA =
 Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
Hash( list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) )] ),
     Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),  Agent(AgentVar( "A" )),
   Key( KeyComposite(list[   TextOrigin("symmetric_key_AB", AgentVar( "A" )) ] ) )] ) ;

  messageFromAToBagentAencryptKeyABhashnonceRBBnonceRAAagentAkeyKeyCompositetextsymmetrickeyABA
 = Message(AgentVar( "A" ),AgentVar( "B" ),
 list[  Agent(AgentVar( "A" )),
 Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
Hash( list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) )] ),
            Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),
Agent(AgentVar( "A" )),
  Key( KeyComposite(list[TextOrigin("symmetric_key_AB", AgentVar( "A" )) ] ) )] )]) ;

  network!send( Message(AgentVar( "A" ),AgentVar( "B" ),
 list[  Agent(AgentVar( "A" )),
 Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
Hash( list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) )] ),
 Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),  Agent(AgentVar( "A" )),
  Key( KeyComposite(list[   TextOrigin("symmetric_key_AB", AgentVar( "A" )) ] ) )] )]) )  ;

  messageFromBToAagentBencryptKeyCompositetextsymmetrickeyABAhashnonceNAA
  = Message(AgentVar( "B" ),AgentVar( "A" ),
 list[  Agent(AgentVar( "B" )),
 Encrypt( KeyComposite(list[   TextOrigin("symmetric_key_AB", AgentVar( "A" )) ] ) , list[
Hash( list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) )] )]) ;

  await( messageFromBToAagentBencryptKeyCompositetextsymmetrickeyABAhashnonceNAAQuery )  ;

  decryptKeyCompositetextsymmetrickeyABAencryptKeyCompositetextsymmetrickeyABAhashnonceNAA =
 Decrypt(   KeyComposite(list[   TextOrigin("symmetric_key_AB", AgentVar( "A" )) ] ) , list[
 Encrypt( KeyComposite(list[   TextOrigin("symmetric_key_AB", AgentVar( "A" )) ] ) , list[
Hash( list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) )] )] ) ] ) ;

  hashnonceNAA =
Hash( list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) )] ) ;

    }
  /* End of run-method */


   Unit receive(ProtocolClause msg){
 if ( testmessageFromBToAagentBnonceRBB (msg) )
 {messageFromBToAagentBnonceRBBQuery = True ;

    }
    else  {
 if ( testmessageFromBToAagentBencryptKeyCompositetextsymmetrickeyABAhashnonceNAA (msg) )
 {messageFromBToAagentBencryptKeyCompositetextsymmetrickeyABAhashnonceNAAQuery = True ;

    }
    else  { skip  ;

      }
  /* End of test method */

    }
  /* End of test method */

 }
  /* End of receive method */

 }
 /* End of class */

 def Bool
testmessageFromAToBagentAencryptKeyABhashnonceRBBnonceRAAagentAkeyKeyCompositetextsymmetrickeyABA
             (ProtocolClause msg)
= case msg {Message( AgentVar( "A" ),   AgentVar( "B" ),

      Cons (  Agent(AgentVar( "A" )),
```

```
      Cons (  Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )),
      Cons (  Hash(
      Cons ( Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ),  Nil
) ) ,
      Cons ( Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),
      Cons (  Agent(AgentVar( "A" )),
      Cons (  Key( KeyComposite(list[   TextOrigin("symmetric_key_AB", AgentVar( "A" )) ] ) ),  Nil
) ) ) ) ) , Nil
) ) ) =>  True ;

    _ => False ;

    } ;



class  Bclass (AgentTerm name, Network network) implements Agent{
Bool
messageFromAToBagentAencryptKeyABhashnonceRBBnonceRAAagentAkeyKeyCompositetextsymmetrickeyABAQuery
 = False  ;

    PayloadElement agentB =  UndefinedElem   ;

    PayloadElement newnonceRBB =  UndefinedElem   ;

    PayloadElement agentA =  UndefinedElem   ;

    ProtocolClause messageFromBToAagentBnonceRBB =  UndefinedClause  ;

    ProtocolClause
messageFromAToBagentAencryptKeyABhashnonceRBBnonceRAAagentAkeyKeyCompositetextsymmetrickeyABA
       =  UndefinedClause   ;

    PayloadElement keyKeyAB =  UndefinedElem   ;

    PayloadElement
decryptKeyABencryptKeyABhashnonceRBBnonceRAAagentAkeyKeyCompositetextsymmetrickeyABA
           =  UndefinedElem   ;

    PayloadElement hashnonceRBB =  UndefinedElem   ;

    PayloadElement nonceRAA =  UndefinedElem   ;

    PayloadElement keyKeyCompositetextsymmetrickeyABA =  UndefinedElem   ;

    PayloadElement textsymmetrickeyABA =  UndefinedElem   ;

    PayloadElement nonceNAA =  UndefinedElem   ;

    PayloadElement hashnonceNAA =  UndefinedElem   ;

    PayloadElement encryptKeyCompositetextsymmetrickeyABAhashnonceNAA
           =  UndefinedElem   ;

    ProtocolClause
messageFromBToAagentBencryptKeyCompositetextsymmetrickeyABAhashnonceNAA
       =  UndefinedClause  ;

      Unit run(){ await(network != null) ;

     agentB =  Agent(AgentVar( "B" )) ;

  newnonceRBB =  New(Nonce( NonceVar ( "RB" ), AgentVar( "B" ) ) ) ;

  agentA =  Agent(AgentVar( "A" )) ;

  messageFromBToAagentBnonceRBB = Message(AgentVar( "B" ),AgentVar( "A" ),
 list[  Agent(AgentVar( "B" )), Nonce( NonceVar ( "RB" ), AgentVar( "B" ) )]) ;

  network!send( Message(AgentVar( "B" ),AgentVar( "A" ),
 list[  Agent(AgentVar( "B" )), Nonce( NonceVar ( "RB" ), AgentVar( "B" ) )]) )  ;
```

```
    messageFromAToBagentAencryptKeyABhashnonceRBBnonceRAAagentAkeyKeyCompositetextsymmetrickeyABA
    = Message(AgentVar( "A" ),AgentVar( "B" ),
   list[  Agent(AgentVar( "A" )),
   Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
  Hash( list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) )] ),
  Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),
Agent(AgentVar( "A" )),
        Key( KeyComposite(list[    TextOrigin("symmetric key AB", AgentVar( "A" )) ] ) )] )]) ;

    await(
  messageFromAToBagentAencryptKeyABhashnonceRBBnonceRAA-
agentAkeyKeyCompositetextsymmetrickeyABAQuery) ;

    keyKeyAB =  Key(KeySymmetric( AgentVar( "A" ),AgentVar( "B" ))) ;

    decryptKeyABencryptKeyABhashnonceRBBnonceRAAagentAkeyKeyCompositetextsymmetrickeyABA =
   Decrypt(  KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
   Encrypt(KeySymmetric( AgentVar( "A" ),AgentVar( "B" )), list[
  Hash( list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) )] ),
    Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ),  Agent(AgentVar( "A" )),
   Key( KeyComposite(list[    TextOrigin("symmetric key AB", AgentVar( "A" )) ] ) )] ) ] ) ;

    hashnonceRBB =
  Hash( list[ Nonce( NonceVar ( "RB" ), AgentVar( "B" ) )] ) ;

    nonceRAA = Nonce( NonceVar ( "RA" ), AgentVar( "A" ) ) ;

    keyKeyCompositetextsymmetrickeyABA =
      Key( KeyComposite(list[    TextOrigin("symmetric key AB", AgentVar( "A" )) ] ) ) ;

    textsymmetrickeyABA =  TextOrigin("symmetric key AB", AgentVar( "A" ))  ;

    nonceNAA = Nonce( NonceVar ( "NA" ), AgentVar( "A" ) ) ;

    hashnonceNAA =
  Hash( list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) )] ) ;

    encryptKeyCompositetextsymmetrickeyABAhashnonceNAA =
   Encrypt( KeyComposite(list[    TextOrigin("symmetric key AB", AgentVar( "A" )) ] ) , list[
  Hash( list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) )] )] ) ;

    messageFromBToAagentBencryptKeyCompositetextsymmetrickeyABAhashnonceNAA
      = Message(AgentVar( "B" ),AgentVar( "A" ),
   list[  Agent(AgentVar( "B" )),
   Encrypt( KeyComposite(list[    TextOrigin("symmetric key AB", AgentVar( "A" )) ] ) , list[
  Hash( list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) )] )]) ;

    network!send( Message(AgentVar( "B" ),AgentVar( "A" ),
   list[  Agent(AgentVar( "B" )),
   Encrypt( KeyComposite(list[    TextOrigin("symmetric key AB", AgentVar( "A" )) ] ) , list[
  Hash( list[ Nonce( NonceVar ( "NA" ), AgentVar( "A" ) )] )]) )  ;

      }
    /* End of run-method */


     Unit receive(ProtocolClause msg){
   if
( testmessageFromAToBagentAencryptKeyABhashnonceRBB-
      nonceRAAagentAkeyKeyCompositetextsymmetrickeyABA (msg) )
   {messageFromAToBagentAencryptKeyABhashnonceRBBnonceRAA-
      agentAkeyKeyCompositetextsymmetrickeyABAQuery = True ;

      }
      else  { skip  ;

        }
    /* End of test method */

   }
    /* End of receive method */
```

```
 }
 /* End of class */


// The protocol: ISO-SYM- Two-Pass non-reversible-functions
{
Agent zombieAgent =  new cog ZombieAgent(AgentName("TestZombie") )  ;

  Network network  ;

  network =  new cog Network(zombieAgent)  ;

  Agent objectA ;

  objectA = new cog Aclass (AgentVar( "A" ) , network) ;

  network!register(AgentVar( "A" ), objectA)  ;

  Agent objectB ;

  objectB = new cog Bclass (AgentVar( "B" ) , network) ;

  network!register(AgentVar( "B" ), objectB)  ;

  }
// end of file
```

Internet Key Exchange version 2 with mac

Note that some of the names are manipulated afterwards in order to make it readable - the strings are significantly longer than can be presented nicely

```
module TestProject;

import * from ABSProtocols;
import * from ExecutionEnvironment;

  def Bool
   testmessageFromBToAtextCryptoOfferIKESABnonceKEYBBnonceBNONCEB (ProtocolClause msg)
 = case msg {Message( AgentVar( "B" ),   AgentVar( "A" ),

       Cons (  TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" )) ,
       Cons ( Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ),
       Cons ( Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),  Nil
  ) ) ) )     => True ;

     _ => False ;

     } ;


  def Bool testmessageFromBToAencryptKeyCompositehashnonceANONCEAnonceBNONCEB
-textCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPS
-KtextCryptoOfferIKESABnonceKEYBBnonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSAB
     (ProtocolClause msg)
 = case msg {Message( AgentVar( "B" ),   AgentVar( "A" ),

       Cons (  Encrypt(KeyComposite (
       Cons (  Hash(
       Cons ( Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
       Cons ( Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
       Cons (  TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
       Cons (  Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
               Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) ),  Nil
  ) ) ) ) ) ,  Nil
  ) ),
       Cons (  Agent(AgentVar( "B" )),
       Cons (  Hash(
       Cons (  Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ),KeyCounterVar( "PSK"  ) )),
       Cons (  TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" )) ,
       Cons ( Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ),
       Cons ( Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
       Cons ( Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),  Nil
  ) ) ) ) ) ) ,
       Cons (  TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "B" )) ,  Nil
  ) ) ) ) ,  Nil
  ) )     => True ;

     _ => False ;

     } ;




 class  Aclass (AgentTerm name, Network network) implements Agent{
 Bool messageFromBToAtextCryptoOfferIKESABnonceKEYBBnonceBNONCEBQuery = False  ;

   Bool
messageFromBToAencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOffer-
IKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPSKtextCryptoOffer-
IKESABnonceKEYBBnonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSABQuery = False  ;

    PayloadElement textCryptoOfferIKESAA =  UndefinedElem   ;

    PayloadElement newnonceKEYAA =  UndefinedElem   ;

    PayloadElement newnonceANONCEA =  UndefinedElem   ;

    PayloadElement agentB =  UndefinedElem   ;

    ProtocolClause messageFromAToBtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA
   =  UndefinedClause  ;
```

```
    ProtocolClause messageFromBToAtextCryptoOfferIKESABnonceKEYBBnonceBNONCEB
    = UndefinedClause ;

    PayloadElement textCryptoOfferIKESAB = UndefinedElem ;

    PayloadElement nonceKEYBB = UndefinedElem ;

    PayloadElement nonceBNONCEB = UndefinedElem ;

    PayloadElement agentA = UndefinedElem ;

    PayloadElement newkeyKeyABPSK = UndefinedElem ;

    PayloadElement
    hashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEAnonceBNONCEB
      = UndefinedElem ;

    PayloadElement textCryptoOfferCHILDSAA = UndefinedElem ;

    PayloadElement keyKeyCompositenonceKEYAAnonceKEYBB
    = UndefinedElem ;

    PayloadElement
    hashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBB
   = UndefinedElem ;

    PayloadElement
keyKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOffer-
IKESAAkeyKeyCompositenonceKEYAAnonceKEYBB = UndefinedElem ;

    PayloadElement encryptKeyCompositehashnonceANONCEAnonceBNONCEB-
textCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentAhash-
keyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEAnonceBNONCEB-
textCryptoOfferCHILDSAA = UndefinedElem ;

    ProtocolClause messageFromAToBencryptKeyCompositehashnonceANONCEA-
nonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBB-
agentAhashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEAnonce-
BNONCEBtextCryptoOfferCHILDSAA = UndefinedClause ;

    ProtocolClause messageFromBToAencryptKeyCompositehashnonceANONCEA-
nonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentB-
hashkeyKeyABPSKtextCryptoOfferIKESABnonceKEYBBnonceANONCEAnonceBNONCEBtext-
CryptoOfferCHILDSAB = UndefinedClause ;

    PayloadElement decryptKeyCompositehashnonceANONCEAnonceBNONCEBtext-
CryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBBencryptKeyComposite-
hashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKeyComposite-
nonceKEYAAnonceKEYBBagentBhashkeyKeyABPSKtextCryptoOfferIKESABnonceKEYBBnonce-
ANONCEAnonceBNONCEBtextCryptoOfferCHILDSAB = UndefinedElem ;

    PayloadElement
hashkeyKeyABPSKtextCryptoOfferIKESABnonceKEYBBnonceANONCEAnonceBNONCEB
= UndefinedElem ;

    PayloadElement textCryptoOfferCHILDSAB = UndefinedElem ;

      Unit run(){ await(network != null) ;

     textCryptoOfferIKESAA = TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ;

   newnonceKEYAA = New(Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ) ) ;

   newnonceANONCEA = New(Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ) ) ;

   agentB = Agent(AgentVar( "B" )) ;

   messageFromAToBtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA =
Message(AgentVar( "A" ),AgentVar( "B" ),
  list[ TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
  Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) )]) ;
```

```
    network!send( Message(AgentVar( "A" ),AgentVar( "B" ),
   list[  TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
   Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
   Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) )]) )  ;

    messageFromBToAtextCryptoOfferIKESABnonceKEYBBnonceBNONCEB =
    Message(AgentVar( "B" ),AgentVar( "A" ),
   list[  TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" )) ,
   Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ),
   Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )]) ;

    await( messageFromBToAtextCryptoOfferIKESABnonceKEYBBnonceBNONCEBQuery )  ;

    textCryptoOfferIKESAB =  TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" ))  ;

    nonceKEYBB = Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ) ;

    nonceBNONCEB = Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ) ;

    agentA =  Agent(AgentVar( "A" )) ;

    newkeyKeyABPSK =  New(Key( KeySymmetricFresh( AgentVar( "A" ),
   AgentVar( "B" ),KeyCounterVar( "PSK"  ) ))) ;

    hashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEAnonceBNONCEB =
 Hash( list[  Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ), KeyCounterVar( "PSK"  ) )),
 TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) , Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
     Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )] ) ;

    textCryptoOfferCHILDSAA =  TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "A" ))  ;

    keyKeyCompositenonceKEYAAnonceKEYBB =
Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
    Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) ) ;

    hashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBB =
 Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
 Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ), TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )),
Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] ) ;

 keyKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBB
 =  Key( KeyComposite(list[
 Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ), TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )),
Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
 Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] ) )] ) ;

    encryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKey-
CompositenonceKEYAAnonceKEYBBagentAhashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonce-
ANONCEAnonceBNONCEBtextCryptoOfferCHILDSAA =
   Encrypt( KeyComposite(list[
 Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
 TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
     Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] ) , list[  Agent(AgentVar( "A" )),
 Hash( list[  Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ),KeyCounterVar( "PSK"  ) )),
TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) , Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ), Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )] ),
     TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "A" )) ] ) ;

    messageFromAToBencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOffer-
IKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentAhashkeyKeyABPSKtextCryptoOffer-
IKESAAnonceKEYAAnonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSAA =
Message(AgentVar( "A" ),AgentVar( "B" ),
   list[
   Encrypt( KeyComposite(list[
 Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
      Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
```

```
TextOrigin("CryptoOffer_IKE_SA", AgentVar("A")),
Key(KeyComposite(list[ Nonce(NonceVar("KEYA")), AgentVar("A")),
Nonce(NonceVar("KEYB"), AgentVar("B"))])), list[ Agent(AgentVar("A")),
 Hash(list[ Key(KeySymmetricFresh(AgentVar("A"),AgentVar("B"),KeyCounterVar("PSK"))),
TextOrigin("CryptoOffer_IKE_SA", AgentVar("A")), Nonce(NonceVar("KEYA"), AgentVar("A")),
Nonce(NonceVar("ANONCE"), AgentVar("A")), Nonce(NonceVar("BNONCE"), AgentVar("B"))]),
TextOrigin("CryptoOffer_CHILD_SA", AgentVar("A")) ])]) ;

   network!send(Message(AgentVar("A"),AgentVar("B"),
  list[
  Encrypt(KeyComposite(list[
 Hash(list[ Nonce(NonceVar("ANONCE")), AgentVar("A")),
   Nonce(NonceVar("BNONCE"), AgentVar("B")),
TextOrigin("CryptoOffer_IKE_SA", AgentVar("A")),
Key(KeyComposite(list[ Nonce(NonceVar("KEYA"), AgentVar("A")),
     Nonce(NonceVar("KEYB"), AgentVar("B"))])])]),
list[ Agent(AgentVar("A")),
 Hash(list[ Key(KeySymmetricFresh(AgentVar("A"),AgentVar("B"),KeyCounterVar("PSK"))),
TextOrigin("CryptoOffer_IKE_SA", AgentVar("A")), Nonce(NonceVar("KEYA"), AgentVar("A")),
Nonce(NonceVar("ANONCE"), AgentVar("A")), Nonce(NonceVar("BNONCE"), AgentVar("B"))]),
TextOrigin("CryptoOffer_CHILD_SA", AgentVar("A")) ])]) ;

   messageFromBToAencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOffer-
IKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPSKtextCryptoOffer-
IKESABnonceKEYBBnonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSAB =
 Message(AgentVar("B"),AgentVar("A"),
  list[
  Encrypt(KeyComposite(list[
 Hash(list[ Nonce(NonceVar("ANONCE"), AgentVar("A")),
                Nonce(NonceVar("BNONCE"), AgentVar("B")),
TextOrigin("CryptoOffer_IKE_SA", AgentVar("A")),
Key(KeyComposite(list[ Nonce(NonceVar("KEYA"), AgentVar("A")),
     Nonce(NonceVar("KEYB"), AgentVar("B"))])]), list[ Agent(AgentVar("B")),
 Hash(list[ Key(KeySymmetricFresh(AgentVar("A"),AgentVar("B"),KeyCounterVar("PSK"))),
TextOrigin("CryptoOffer_IKE_SA", AgentVar("B")), Nonce(NonceVar("KEYB"), AgentVar("B")),
Nonce(NonceVar("ANONCE"), AgentVar("A")), Nonce(NonceVar("BNONCE"), AgentVar("B"))]),
TextOrigin("CryptoOffer_CHILD_SA", AgentVar("B")) ])]) ;

   await( messageFromBToAencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOffer-
IKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPSKtextCryptoOfferIKESABnonce-
KEYBBnonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSABQuery ) ;

   decryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKeyComposite-
nonceKEYAAnonceKEYBBencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOffer-
IKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPSKtextCryptoOfferIKESAB-
nonceKEYBBnonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSAB =
  Decrypt(   KeyComposite(list[
 Hash(list[ Nonce(NonceVar("ANONCE"), AgentVar("A")),
     Nonce(NonceVar("BNONCE"), AgentVar("B")),
TextOrigin("CryptoOffer_IKE_SA", AgentVar("A")),
 Key(KeyComposite(list[ Nonce(NonceVar("KEYA"), AgentVar("A")),
   Nonce(NonceVar("KEYB"), AgentVar("B"))])])]), list[
  Encrypt(KeyComposite(list[
 Hash(list[ Nonce(NonceVar("ANONCE"), AgentVar("A")),
  Nonce(NonceVar("BNONCE"), AgentVar("B")),
TextOrigin("CryptoOffer_IKE_SA", AgentVar("A")),
Key(KeyComposite(list[ Nonce(NonceVar("KEYA"), AgentVar("A")),
Nonce(NonceVar("KEYB"), AgentVar("B"))])])]),
list[ Agent(AgentVar("B")),
 Hash(list[ Key(KeySymmetricFresh(AgentVar("A"),AgentVar("B"),KeyCounterVar("PSK"))),
TextOrigin("CryptoOffer_IKE_SA", AgentVar("B")), Nonce(NonceVar("KEYB"), AgentVar("B")),
Nonce(NonceVar("ANONCE"), AgentVar("A")),
Nonce(NonceVar("BNONCE"), AgentVar("B"))]),
TextOrigin("CryptoOffer_CHILD_SA", AgentVar("B")) ])]) ;

   hashkeyKeyABPSKtextCryptoOfferIKESABnonceKEYBBnonceANONCEAnonceBNONCEB =
 Hash(list[
Key(KeySymmetricFresh(AgentVar("A"),AgentVar("B"), KeyCounterVar("PSK"))),
TextOrigin("CryptoOffer_IKE_SA", AgentVar("B")), Nonce(NonceVar("KEYB"), AgentVar("B")),
Nonce(NonceVar("ANONCE"), AgentVar("A")),
   Nonce(NonceVar("BNONCE"), AgentVar("B"))]) ;
```

```
   textCryptoOfferCHILDSAB =  TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "B" ))  ;

     }
   /* End of run-method */


    Unit receive(ProtocolClause msg){
  if ( testmessageFromBToAtextCryptoOfferIKESABnonceKEYBBnonceBNONCEB (msg) )
  {messageFromBToAtextCryptoOfferIKESABnonceKEYBBnonceBNONCEBQuery = True ;

     }
     else  {
  if (
testmessageFromBToAencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAA-
keyKeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPSKtextCryptoOfferIKESAB-
nonceKEYBBnonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSAB (msg) )
  {messageFromBToAencryptKeyCompositehashnonceANONCEAnonceBNONCEB-
textCryptoOfferIKESAAkey
KeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPSKtextCryptoOfferIKESAB-
nonceKEYBBnonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSABQuery = True ;

     }
     else  { skip  ;

       }
   /* End of test method */

     }
   /* End of test method */

  }
   /* End of receive method */

  }
  /* End of class */

  def Bool testmessageFromAToBtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA (ProtocolClause msg)
 = case msg {Message( AgentVar( "A" ),   AgentVar( "B" ),

       Cons (  TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
       Cons ( Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
       Cons ( Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),  Nil
  )  )  )  )     => True ;

     _ => False ;

     }  ;


  def Bool
testmessageFromAToBencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAA-
keyKeyCompositenonceKEYAAnonceKEYBBagentAhashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAA-
nonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSAA (ProtocolClause msg)
 = case msg {Message( AgentVar( "A" ),   AgentVar( "B" ),

       Cons (  Encrypt(KeyComposite (
       Cons (  Hash(
       Cons ( Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
       Cons ( Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
       Cons (  TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
       Cons (  Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
               Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) ),  Nil
  )  )  )  )  ) ,  Nil
  ) ),
       Cons (  Agent(AgentVar( "A" )),
       Cons (  Hash(
       Cons (  Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ),KeyCounterVar( "PSK"  ) )),
       Cons (  TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
       Cons ( Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
       Cons ( Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
       Cons ( Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),  Nil
  )  )  )  )  )  ) ,
```

```
        Cons (  TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "A" )) ,  Nil
  ) ) ) ) ,  Nil
  ) )      => True ;

     _ => False ;

     } ;



 class  Bclass (AgentTerm name, Network network) implements Agent{
 Bool messageFromAToBtextCryptoOfferIKESAAnonceKEYAAnonceANONCEAQuery = False  ;

   Bool messageFromAToBencryptKeyCompositehashnonceANONCEAnonceBNONCEB-
textCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentAhash-
keyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonce-
  ANONCEAnonceBNONCEBtextCryptoOfferCHILDSAAQuery = False  ;

    ProtocolClause messageFromAToBtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA =
UndefinedClause   ;

    PayloadElement agentA =  UndefinedElem   ;

    PayloadElement textCryptoOfferIKESAA =  UndefinedElem   ;

    PayloadElement nonceKEYAA =  UndefinedElem   ;

    PayloadElement nonceANONCEA =  UndefinedElem   ;

    PayloadElement textCryptoOfferIKESAB =  UndefinedElem   ;

    PayloadElement newnonceKEYBB =  UndefinedElem   ;

    PayloadElement newnonceBNONCEB =  UndefinedElem   ;

    ProtocolClause messageFromBToAtextCryptoOfferIKESABnonceKEYBBnonceBNONCEB
=  UndefinedClause  ;

    ProtocolClause messageFromAToBencryptKeyCompositehashnonceANONCEA-
nonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBB-
agentAhashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA-
nonceBNONCEBtextCryptoOfferCHILDSAA =  UndefinedClause   ;

    PayloadElement keyKeyCompositenonceKEYAAnonceKEYBB
    =  UndefinedElem   ;

    PayloadElement
hashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBB
                =  UndefinedElem   ;

    PayloadElement
keyKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBB
    =  UndefinedElem   ;

    PayloadElement
 decryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkey-
KeyCompositenonceKEYAAnonceKEYBBencryptKeyCompositehashnonceANONCEA-
nonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBB-
agentAhashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA-
nonceBNONCEBtextCryptoOfferCHILDSAA =  UndefinedElem   ;

    PayloadElement
hashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEAnonceBNONCEB
            =  UndefinedElem   ;

    PayloadElement textCryptoOfferCHILDSAA =  UndefinedElem   ;

    PayloadElement agentB =  UndefinedElem   ;

    PayloadElement newkeyKeyABPSK =  UndefinedElem   ;
```

```
    PayloadElement
hashkeyKeyABPSKtextCryptoOfferIKESABnonceKEYBBnonceANONCEAnonceBNONCEB
                = UndefinedElem ;

    PayloadElement textCryptoOfferCHILDSAB = UndefinedElem ;

    PayloadElement encryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkey-
KeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPSKtextCryptoOfferIKESABnonceKEYBBnonce-
    ANONCEAnonceBNONCEBtextCryptoOfferCHILDSAB = UndefinedElem ;

    ProtocolClause
messageFromBToAencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAA-
keyKeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPSKtextCryptoOfferIKESAB
nonceKEYBBnonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSAB = UndefinedClause ;

      Unit run(){ await(network != null) ;

     messageFromAToBtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA
     = Message(AgentVar( "A" ),AgentVar( "B" ),
  list[ TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
       Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) )]) ;

   await( messageFromAToBtextCryptoOfferIKESAAnonceKEYAAnonceANONCEAQuery ) ;

   agentA = Agent(AgentVar( "A" )) ;

   textCryptoOfferIKESAA = TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ;

   nonceKEYAA = Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ) ;

   nonceANONCEA = Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ) ;

   textCryptoOfferIKESAB = TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" )) ;

   newnonceKEYBB = New(Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ) ) ;

   newnonceBNONCEB = New(Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ) ) ;

   messageFromBToAtextCryptoOfferIKESABnonceKEYBBnonceBNONCEB =
        Message(AgentVar( "B" ),AgentVar( "A" ),
  list[ TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" )),
                                Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ),
Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )]) ;

   network!send( Message(AgentVar( "B" ),AgentVar( "A" ),
  list[ TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" )) ,
                                Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ),
Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )]) ) ;

   messageFromAToBencryptKeyCompositehashnonceANONCEAnonceBNONCEB-
textCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentA-
hashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA-
nonceBNONCEBtextCryptoOfferCHILDSAA = Message(AgentVar( "A" ),AgentVar( "B" ),
  list[
  Encrypt( KeyComposite(list[
 Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
   Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
       Key( KeyComposite(list[ Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] )] ) , list[ Agent(AgentVar( "A" )),
 Hash( list[ Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ),KeyCounterVar( "PSK" ) )),
 TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) , Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ), Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )] ),
TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "A" )) ] )]) ;

   await( messageFromAToBencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAA-
keyKeyCompositenonceKEYAAnonceKEYBBagentAhashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA-
nonceBNONCEBtextCryptoOfferCHILDSAAQuery ) ;

   keyKeyCompositenonceKEYAAnonceKEYBB =
 Key( KeyComposite(list[ Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
```

```
      Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) ) ;

   hashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBB =
 Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
   Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
     Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] ) ;

   keyKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBB =
Key( KeyComposite(list[ Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),  TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
     Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] )] ) ) ;

   decryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKeyComposite-
nonceKEYAAnonceKEYBBencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAA-
keyKeyCompositenonceKEYAAnonceKEYBBagentAhashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAA-
nonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSAA =
  Decrypt(   KeyComposite(list[
 Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
                 Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
    Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
 Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] )] ) , list[
  Encrypt( KeyComposite(list[
 Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
             Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
 TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
     Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] )] ) , list[  Agent(AgentVar( "A" )),
 Hash( list[  Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ),KeyCounterVar( "PSK"  ) )),
TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) , Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ), Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )] ),
TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "A" )) ] ) ] ) ;

   hashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEAnonceBNONCEB =
 Hash( list[  Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ),KeyCounterVar( "PSK"  ) )),
TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) , Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
            Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )] ) ;

   textCryptoOfferCHILDSAA =  TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "A" ))  ;

   agentB =  Agent(AgentVar( "B" )) ;

   newkeyKeyABPSK =  New(Key( KeySymmetricFresh( AgentVar( "A" ),
                      AgentVar( "B" ),KeyCounterVar( "PSK"  ) ))) ;

   hashkeyKeyABPSKtextCryptoOfferIKESABnonceKEYBBnonceANONCEAnonceBNONCEB =
 Hash( list[  Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ),KeyCounterVar( "PSK"  ) )),
TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" )) , Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ),
Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
                       Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )] ) ;

   textCryptoOfferCHILDSAB =  TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "B" ))  ;

   encryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAA-
keyKeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPSKtextCryptoOfferIKESAB-
nonceKEYBBnonceANONCEAnonceBNONCEBtextCryptoOfferCHILDSAB =
  Encrypt( KeyComposite(list[
 Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
   Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
 Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ),
    AgentVar( "A" ) ), Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] ) ,
 list[  Agent(AgentVar( "B" )),
   Hash( list[  Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ),KeyCounterVar( "PSK"  ) )),
 TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" )) ,
   Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ), Nonce( NonceVar ( "ANONCE" ),
AgentVar( "A" ) ), Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )] ),
            TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "B" )) ] ) ;
```

```
    messageFromBToAencryptKeyCompositehashnonceANONCEAnonceBNONCEBtext-
CryptoOfferIKESAAkeyKeyCompositenonceKEYAAnonceKEYBBagentBhashkeyKeyABPSK-
textCryptoOfferIKESABnonceKEYBBnonceANONCEAnonceBNONCE-
BtextCryptoOfferCHILDSAB = Message(AgentVar( "B" ),AgentVar( "A" ),
  list[
  Encrypt( KeyComposite(list[
 Hash( list[ Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ),
           Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
 TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
      Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] ) , list[  Agent(AgentVar( "B" )),
 Hash( list[  Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ),KeyCounterVar( "PSK"  ) )),
 TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" )) , Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ),
Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ), Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )] ),
TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "B" )) ] )]) ;

   network!send( Message(AgentVar( "B" ),AgentVar( "A" ),
  list[
  Encrypt( KeyComposite(list[
 Hash( list[ Nonce( NonceVar ( "ANONCE" ),
             AgentVar( "A" ) ), Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) ),
TextOrigin("CryptoOffer_IKE_SA", AgentVar( "A" )) ,
   Key( KeyComposite(list[  Nonce( NonceVar ( "KEYA" ), AgentVar( "A" ) ),
Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) )] ) )] )] ) , list[  Agent(AgentVar( "B" )),
 Hash( list[  Key( KeySymmetricFresh( AgentVar( "A" ),AgentVar( "B" ),KeyCounterVar( "PSK"  ) )),
TextOrigin("CryptoOffer_IKE_SA", AgentVar( "B" )) , Nonce( NonceVar ( "KEYB" ), AgentVar( "B" ) ),
Nonce( NonceVar ( "ANONCE" ), AgentVar( "A" ) ), Nonce( NonceVar ( "BNONCE" ), AgentVar( "B" ) )] ),
 TextOrigin("CryptoOffer_CHILD_SA", AgentVar( "B" )) ] )]) )  ;

     }
   /* End of run-method */


    Unit receive(ProtocolClause msg){
  if ( testmessageFromAToBtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA (msg) )
  {messageFromAToBtextCryptoOfferIKESAAnonceKEYAAnonceANONCEAQuery = True ;

     }
     else  {
  if ( testmessageFromAToBencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAA-
keyKeyCompositenonceKEYAAnonceKEYBBagentAhashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA-
nonceBNONCEBtextCryptoOfferCHILDSAA (msg) )
  {messageFromAToBencryptKeyCompositehashnonceANONCEAnonceBNONCEBtextCryptoOfferIKESAAkeyKey-
CompositenonceKEYAAnonceKEYBBagentAhashkeyKeyABPSKtextCryptoOfferIKESAAnonceKEYAAnonceANONCEA-
nonceBNONCEBtextCryptoOfferCHILDSAAQuery = True ;

     }
     else  { skip  ;

       }
   /* End of test method */

     }
   /* End of test method */

 }
  /* End of receive method */

 }
 /* End of class */



// The protocol: IKE version 2 mac
{
Agent zombieAgent =  new cog ZombieAgent(AgentName("TestZombie") )  ;

  Network network  ;

  network =  new cog Network(zombieAgent)  ;
```

```
  Agent objectA ;
  objectA = new cog Aclass (AgentVar( "A" ) , network) ;
  network!register(AgentVar( "A" ), objectA)  ;
  Agent objectB ;
  objectB = new cog Bclass (AgentVar( "B" ) , network) ;
  network!register(AgentVar( "B" ), objectB)  ;
  }
// end of file
```