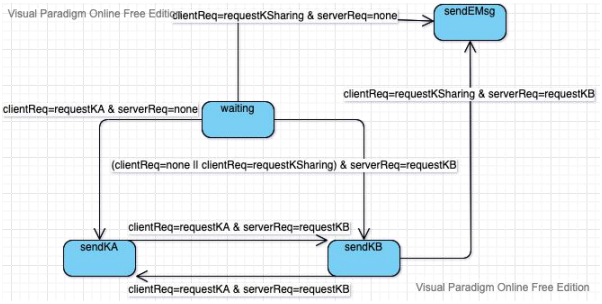
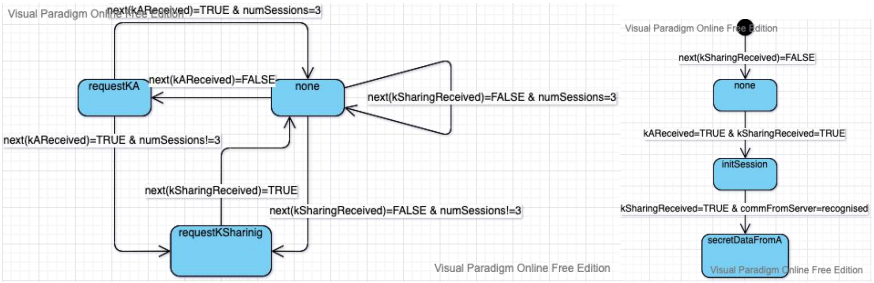
形式化方法实例分析

分析目标：使用 NuSMV 模拟 Kerberos 协议的中间人攻击。

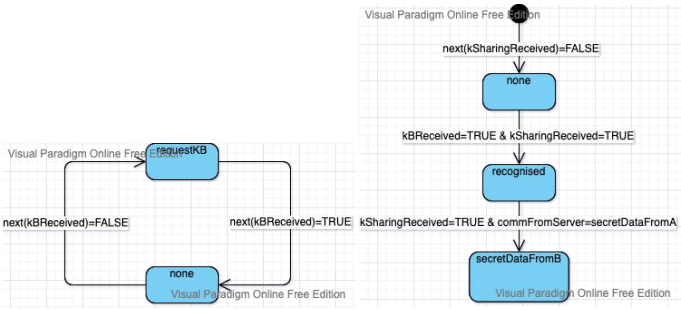
分别对验证服务器 (authServer)、客户端 (client)、服务端 (server)、中间人 (midman) 构造有限状态机。验证服务器的有限状态机如下，主要用来接受客户端和服务端的请求，并对请求进行处理：



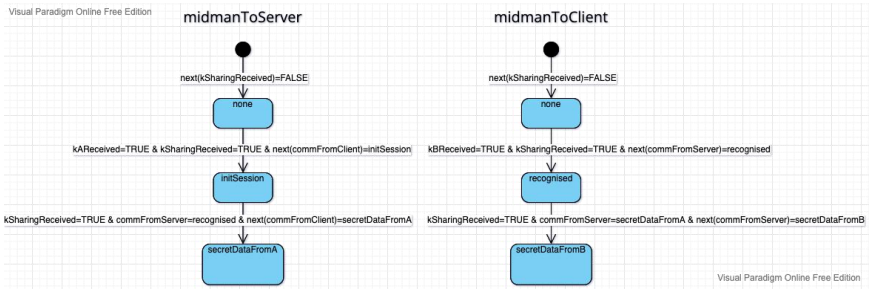
在客户端（client）中，可以分别向 authServer 申请 client 的密钥以及共享密钥，并且设定 session 的最大个数为 3 个，超出后不能够继续申请密钥来创建会话。在接收到密钥后，可以初始化当前会话，如果 server 传来 recognised 消息，可以向 server 传输加密消息：



在服务端（server）中，可以向 authServer 申请 server 的密钥，用于和 client 之间的会话通讯。在接收到密钥后，如果 client 传来共享密钥，则向 client 发送 recognised 消息，接收到 client 的加密消息后，server 会向 client 回复加密消息：



在中间人模块（midman）中，midmanToServer 和 midmanToClient 两个状态机分别表示中间人劫持了 client 发送的消息以及 server 发送的消息。在相同的状态下，中间人可以分别伪造相应的 secretDataFromA 和 secretDataFromB，并发送给双方。宏观来讲，当 client 或 server 接收到加密消息并回复后，说明 client 或 server 被伪造了：



在 main 模块中分别创建各个模块的进程，使用计算树逻辑（CTL Specification）对中间人攻击流程进行校验。存在一种情况，当中间人接收到共享密钥后，client 必定会受到攻击（接收到被篡改的信息）；为了输出状态变化的路径，修改 CTLSPEC 为：在所有中间人接收到共享密钥后的情况下，client 都不会被攻击。NuSMV 输出的攻击路径如下：

```
-- specification EG (m.kSharingReceived = TRUE -> AF m.bAisCompromised = TRUE)  is true
```

```
-- specification AG (m.kSharingReceived = TRUE -> EF m.bAisCompromised = FALSE)  is false
```

```
-> State: 1.1 <-
  t.state = waiting
  c.session = none
  c.authReq = none
  c.serverReq = none
  c.kARecieved = FALSE
  c.kSharingReceived = FALSE
  c.duration = 0
  c.numSessions = 0
  s.session = none
  s.authReq = none
  s.clientReq = none
  s.kBRecieved = FALSE
  s.kSharingReceived = FALSE
  s.duration = 0
  s.numSessions = 0
  m.kAReceived = FALSE
  m.kBReceived = FALSE
  m.kSharingReceived = FALSE
  m.bAisCompromised = FALSE
  m.bBisCompromised = FALSE
  m.midmanToClient = none
  m.midmanToServer = none
  m.duration = 0
  m.session = none
-> Input: 1.2 <-
  _process_selector_ = c
  running = FALSE
  m.running = FALSE
  s.running = FALSE
  c.running = TRUE
  t.running = FALSE
-> State: 1.2 <-
  c.authReq = requestKA
-> Input: 1.3 <-
  _process_selector_ = t
  c.running = FALSE
  t.running = TRUE
-> State: 1.3 <-
  t.state = sendKA
-> Input: 1.4 <-
  _process_selector_ = m
  m.running = TRUE
  t.running = FALSE
-> State: 1.4 <-
  m.kAReceived = TRUE
-> Input: 1.5 <-
  _process_selector_ = c
  m.running = FALSE
  c.running = TRUE
-> State: 1.5 <-
  c.authReq = requestKSharing
  c.kARecieved = TRUE
-> Input: 1.6 <-
  _process_selector_ = t
  c.running = FALSE
  t.running = TRUE
-> State: 1.6 <-
  t.state = waiting
-> Input: 1.7 <-
-> State: 1.7 <-
  t.state = sendEncryptedMsg
-> Input: 1.8 <-
  _process_selector_ = s
  s.running = TRUE
  t.running = FALSE
-> State: 1.8 <-
  s.authReq = requestKB
-> Input: 1.9 <-
  _process_selector_ = m
  m.running = TRUE
  s.running = FALSE
-> State: 1.9 <-
  m.kSharingReceived = TRUE
-> Input: 1.10 <-
  _process_selector_ = c
  m.running = FALSE
  c.running = TRUE
-> State: 1.10 <-
  c.authReq = none
  c.kSharingReceived = TRUE
-> Input: 1.10 <-
  _process_selector_ = c
  m.running = FALSE
  c.running = TRUE
-> State: 1.10 <-
  c.authReq = none
  c.kSharingReceived = TRUE
-> Input: 1.11 <-
  _process_selector_ = t
  c.running = FALSE
  t.running = TRUE
-> State: 1.11 <-
  t.state = waiting
-> Input: 1.12 <-
-> State: 1.12 <-
  t.state = sendKB
-> Input: 1.13 <-
  _process_selector_ = c
  c.running = TRUE
  t.running = FALSE
-> State: 1.13 <-
  c.serverReq = initSession
-> Input: 1.14 <-
  _process_selector_ = m
  m.running = TRUE
  c.running = FALSE
-> State: 1.14 <-
  m.kBReceived = TRUE
  m.midmanToServer = initSession
-> Input: 1.15 <-
  _process_selector_ = s
  m.running = FALSE
  s.running = TRUE
-> State: 1.15 <-
  s.authReq = none
  s.kBRecieved = TRUE
-> Input: 1.16 <-
-> State: 1.16 <-
  s.kSharingReceived = TRUE
-> Input: 1.17 <-
-> State: 1.17 <-
  s.clientReq = recognised
-> Input: 1.18 <-
  _process_selector_ = m
  m.running = TRUE
  s.running = FALSE
-> State: 1.18 <-
  m.kBReceived = FALSE
  m.midmanToClient = recognised
-> Input: 1.19 <-
  _process_selector_ = c
  m.running = FALSE
  c.running = TRUE
-> State: 1.19 <-
  c.session = active
  c.serverReq = secretDataFromA
  c.numSessions = 1
-> Input: 1.20 <-
  _process_selector_ = m
  m.running = TRUE
  c.running = FALSE
-> State: 1.20 <-
  m.bAisCompromised = TRUE
  m.midmanToServer = secretDataFromA
  m.session = active
```

代码：

```
-- Trent (authentic server)
MODULE authServer(clientReq, serverReq)
VAR
  state : {waiting, sendKA, sendKB, sendEncryptedMsg};
ASSIGN
  -- authentic server's state
  init(state) := waiting; -- waiting first
  next(state) := case
    state = waiting & clientReq = requestKA & serverReq = requestKB : {sendKA, sendKB}; -- send
      state = sendKA & clientReq = requestKA & serverReq = requestKB : {sendKB}; -- if already s
      state = sendKB & clientReq = requestKA & serverReq = requestKB : {sendKA}; -- if already s
      state = sendKB & clientReq = requestKSharing & serverReq = requestKB : {sendEncryptedMsg};
      state = waiting & clientReq = requestKSharing & serverReq = requestKB : {sendKB}; -- if A
      state = waiting & clientReq = requestKA & serverReq = none : {sendKA}; -- if A send reques
      state = waiting & clientReq = requestKSharing & serverReq = none : {sendEncryptedMsg}; --
      state = waiting & clientReq = none & serverReq = requestKB : {sendKB}; -- if B send reques
      TRUE : waiting;
  esac;
FAIRNESS running
```

```
-- Alice (client)
MODULE client(authState, commFromServer)
VAR
  session : {none, active};
  authReq : {none, requestKA, requestKSharing};
  serverReq : {none, initSession, secretDataFromA};
  kARecieved : boolean; -- if received A's key or not
  kSharingReceived : boolean; -- if received sharing key or not
  duration : {0, 1, 2, 3, 4, 5}; -- max(duration) = 5
  numSessions : {0, 1, 2, 3}; -- max(sessions) = 3
ASSIGN
  -- authentic request
  init(authReq) := none;
  next(authReq) := case
    authReq = none & next(kARecieved) = FALSE : requestKA; -- if no request & no receive
    authReq = none & next(kSharingReceived) = FALSE & numSessions != 3 : requestKSharing;
    authReq = none & next(kSharingReceived) = FALSE & numSessions = 3 : none; -- if no re
    authReq = requestKA & next(kARecieved) = TRUE & numSessions != 3 : requestKSharing; --
    authReq = requestKA & next(kARecieved) = TRUE & numSessions = 3 : none; -- if request
    authReq = requestKSharing & next(kSharingReceived) = TRUE : none; -- if request shari
    TRUE : authReq;
  esac;
```

```
-- if received A's key or not
init(kARecieved) := FALSE;
next(kARecieved) := case
  authReq = requestKA & authState = sendKA : TRUE; -- if request A's key & authentic server send A's
    TRUE : kARecieved;
esac;
-- if received sharing key or not
init(kSharingReceived) := FALSE;
next(kSharingReceived) := case
  kARecieved = TRUE & authState = sendEncryptedMsg : TRUE; -- if received A's key & authentic server
  next(duration) = 5 : FALSE;
  TRUE : kSharingReceived;
esac;
-- server request
init(serverReq) := none;
next(serverReq) := case
  serverReq = none & kARecieved = TRUE & kSharingReceived = TRUE : initSession; -- if server request
  serverReq = initSession & kSharingReceived = TRUE & commFromServer = recognised : secretDataFromA;
  next(kSharingReceived) = FALSE : none;
  TRUE : serverReq;
esac;
```

```
-- duration
init(duration) := 0;
next(duration) := case
  session = none : 0;
  session = active & duration = 5 : 5; -- max(duration) = 5
  session = active : duration + 1;
  TRUE : duration;
esac;
-- session
init(session) := none;
next(session) := case
  next(duration) = 5 : none; -- if duration = 5, no more session
  kSharingReceived = TRUE & commFromServer = recognised : active;
  TRUE : session;
esac;
-- numSessions
init(numSessions) := 0;
next(numSessions) := case
  session = none & next(session) = active & numSessions = 3 : 3;
  session = none & next(session) = active : numSessions + 1; -- i
  TRUE : numSessions;
esac;
FAIRNESS running
```

```
-- Bob (server)
MODULE server(authState, commFromClient)
VAR
  -- 2 states for session
  session : {none, active};
  -- 2 states for authentic request
  authReq : {none, requestKB};
  -- 3 states for client request
  clientReq : {none, recognised, secretDataFromB};
  kBRecieved : boolean; -- if B's key received or not
  kSharingReceived : boolean; -- if sharing key received or
  duration : {0, 1, 2, 3, 4, 5}; -- max(duration) = 5
  numSessions : {0, 1, 2, 3}; -- max(sessions) = 3
ASSIGN
  -- authentic request
  init(authReq) := none;
  next(authReq) := case
    authReq = none & next(kBRecieved) = FALSE : requestKB;
    authReq = requestKB & next(kBRecieved) = TRUE : none;
    TRUE : authReq;
  esac;
  -- if B's key received or not
  init(kBRecieved) := FALSE;
  next(kBRecieved) := case
    authReq = requestKB & authState = sendKB : TRUE; -- if
      TRUE : kBRecieved;
  esac;
```

```
-- Man in the middle
MODULE midman(authState, commFromClient, commFromServer)
VAR
  kAReceived : boolean;
  kBReceived : boolean;
  kSharingReceived : boolean;
  bAisCompromised : boolean;
  bBisCompromised : boolean;
  midmanToClient: {none, recognised, secretDataFromB};
  midmanToServer: {none, initSession, secretDataFromA};
  duration: {0, 1, 2, 3, 4, 5};
  session: {none, active};
ASSIGN
  -- bKAS is A's secret key, needed to decrypt the sessi
  -- midman simply listens for it and stores it when fou
  init(kAReceived) := FALSE;
  next(kAReceived) := case
    authState = sendKA : {TRUE, FALSE};
    TRUE : kAReceived;
  esac;
  -- bKBS is B's secret key, needed to decrypt the sessi
  -- midman simply listens for it and stores it when fou
  init(kBReceived) := FALSE;
  next(kBReceived) := case
    authState = sendKB : {TRUE, FALSE};
    TRUE : kBReceived;
  esac;
```

```
-- bAisCompromised is a flag to show that we have
init(bAisCompromised) := FALSE;
next(bAisCompromised) := case
  next(commFromClient) = secretDataFromA : TRUE;
  TRUE : bAisCompromised;
esac;
-- bBisCompromised is a flag to show that we have
init(bBisCompromised) := FALSE;
next(bBisCompromised) := case
  next(commFromServer) = secretDataFromB : TRUE;
  TRUE : bBisCompromised;
esac;
```

```
-- if sharing key received of not
init(kSharingReceived) := FALSE;
next(kSharingReceived) := case
  kBRecieved = TRUE & commFromClient = initSession : TRUE; -- if received B's key & commnication from A is ini
  next(duration) = 5 : FALSE; -- if duration reach the max, stop
  TRUE : kSharingReceived;
esac;
-- client request
init(clientReq) := none;
next(clientReq) := case
  clientReq = none & kBRecieved = TRUE & kSharingReceived = TRUE : recognised; -- if A not request & received
  clientReq = recognised & next(kSharingReceived) = TRUE & commFromClient = secretDataFromA : secretDataFromB;
  next(kSharingReceived) = FALSE : none;
  TRUE : clientReq;
esac;
```

```
-- kerberos main loop
MODULE main
VAR
  t : process authServer(c.authReq, s.authReq);
  c : process client(t.state, m.midmanToClient); -- midman hijack the request
  s : process server(t.state, m.midmanToServer); -- midman hijack the request
  m : process midman(t.state, c.serverReq, s.clientReq);
-- if midman hijacks sharing key, midman can pretend as A
CTLSPEC EG (m.kSharingReceived = TRUE -> AF m.bAisCompromised = TRUE)
CTLSPEC AG (m.kSharingReceived = TRUE -> EF m.bAisCompromised = FALSE)
```

```
init(kSharingReceived) := FALSE;
next(kSharingReceived) := case
  kAReceived = TRUE & authState = sendEncryptedMsg : {TRUE, FALSE};
  next(duration) = 5 : FALSE;
  TRUE : kSharingReceived;
esac;
-- midmanToClient forwards messages received from Server as long as we
-- have the session key to decrypt them.
init(midmanToClient) := none;
next(midmanToClient) := case
  next(commFromServer) = recognised & kBReceived = TRUE & kSharingReceived = TRUE : recognised;
  next(commFromServer) = secretDataFromB & kBReceived = TRUE & kSharingReceived = TRUE : secretDataFromB;
  next(kSharingReceived) = FALSE : none;
  TRUE : midmanToClient;
esac;
-- midmanToClient forwards messages received from Client as long as we
-- have the session key to decrypt them.
init(midmanToServer) := none;
next(midmanToServer) := case
  next(commFromClient) = initSession & kAReceived = TRUE & kSharingReceived = TRUE : initSession;
  next(commFromClient) = secretDataFromA & kAReceived = TRUE & kSharingReceived = TRUE : secretDataFromA;
  next(kSharingReceived) = FALSE : none;
  TRUE : midmanToServer;
esac;
```