International Journal of Security and Its Applications
Vol.9, No.7 (2015), pp.307-316
http://dx.doi.org/10.14257/ijsia.2015.9.7.28

NSPK Protocol Security Model Checking System Builder

Wang Yan, Liu Ying

Information Engineering College, Zhongzhou University, Zhengzhou 450044;
PLA Information Engineering University , Zhengzhou 450000
zzdxwyan@126.com

Abstract

Cryptographic protocols are based cryptosystems based on interactive communication protocol, running on computer communication networks or distributed systems, by means of cryptographic algorithms to achieve key distribution, authentication and other purposes. In the safety analysis, model checking technology has a high degree of automation, can provide advantages such as counter-examples. The key application of model checking techniques cryptographic protocol analysis is that the full optimization model checking tools for modeling cryptographic protocols, especially for modeling intruders. Using symbolic model checking techniques and model checking tool UPPAAL automatic machine modeling, it is beneficial for the security of cryptographic protocols for analysis testing.

Keywords: *cryptographic protocols, formal analysis, model checking, UPPAAL*

Password protocol is built on the basis of cryptosystem interactive communication protocol, running on the computer communications network or distributed system, by means of a cryptographic algorithm to achieve key distribution, authentication and other purposes. Currently, cryptographic protocols have been widely used in computer networks and distributed systems, but security analysis of cryptographic protocols is still a problem. With other cryptographic protocol analysis technology compared to the safety analysis, model checking technology has a high degree of automation, can provide counter-examples, and other advantages. The key application of model checking techniques cryptographic protocol analysis is that the full optimization model checking tools for modeling cryptographic protocols, especially for modeling intruders. Therefore , starting on modeling cryptographic protocols exploration and research has important significance from model checking purposes.

NS protocol often (especially NSPK protocol) as an experimental model to detect objects of various model checking tools for password protocol model checking experiments . We also model checking NSPK agreement as subjects of our application system model construction methods.

First, the overall design of the detection system model NSPK protocol model are as follows .

We designed three main template Sender, Receiver, Network and Intruder detection system Models NSPK agreement. Sender authentication template is modeled on the initiator, requires two parameters: its own identity signs myId, and set each other's identity logo yourId, because the system definition needs to directly specify their intended responder certification. Receiver template modeling responder certification only requires one parameter: their identity mark myId, because they do not specify the authentication initiator system definition, but determined dynamically authenticate the initiator in the system is running. Network parameters template for their own identity mark myId.

Figure 1 shows the system model is defined constants, types, variables and channels.

ISSN: 1738-9976 IJSIA

```
const int MAX_HOST_NUM = 2;
const int MAX_NONCE_NUM =10;
const int MSG_TYPE_NUM = 3;
typedef int[0,MAX_HOST_NUM+1] HostId;
typedef int[0,MAX_NONCE_NUM+1] Nonce;
typedef int[1,MSG_TYPE_NUM] MsgType;
const HostId NullHost = 0;
const Nonce NullNonce = 0;
const Nonce INonce =    MAX_NONCE_NUM+1;
const HostId A = 0, HostId B = 1, I = MAX_HOST_NUM ;
const MsgType Msg1 = 1, Msg2 = 2, Msg3 = 3;
typedef struct{
      MsgType msg_type;
      HostId creator;
      HostId sender;
      HostId receiver;
      Nonce nonce1;
      Nonce nonce2;
}Msg;
Msg curMsg[MAX_HOST_NUM];
chan msg[MAX_HOST_NUM];
Nonce curNonce = 0;
const int MAX_S_MSG = 10;
```

Statement of Figure 1 NSPK Protocol System Model for Model Checking

System model operating parameters including MAX_HOST_NUM, MAX_NONCE_NUM and MAX_S_MSG. MAX_HOST_NUM constant defined as 2 , except for the limit system model has two main foreign invaders : Certified initiator and responder certification . Constant MAX_NONCE_NUM authentication process performed for frequency control . MAX_S_MSG used to limit the number of intruders messages are stored. After two constants values can be adjusted in the model checking process, the greater the value of their time and space required for the model to detect the greater the demand , however , found that the higher the likelihood of problems.

Each body (including the intruder) has an ID, type HostId, defining statement : typedef int [0, MAX_HOST_NUM + 1] HostId. 1..MAX_HOST_NUM each agreement is subject ID, MAX_HOST_NUM + 1 is an intruder ID, null HostId NullHost type variable is 0. Define constants A = 1, B = 2, I = MAX_HOST_NUM + 1, is easy to use system definition .

Statement typedef int [0, MAX_NONCE_NUM + 1] Nonce; fresh value defines the type Nonce. Intruder without replacing its Nonce value , fixed MAX_NONCE_NUM + 1, because the meaning of the fresh value is by its randomness ensure that as a temporary status symbol , which for the intruder is obviously unimportant . Therefore , the definition of constants INonce = MAX_NONCE_NUM + 1. Null NullNonce Nonce type variable has a value of 0 .

By global variables curNonce randomly generated to simulate the behavior of the main agreement fresh values. Specific way: when the system is initialized curNonce value is set to 0 ; when an entity is required to generate a fresh value , increase the value of curNonce 1 ,

and an increase in the value of the value of their own after a fresh next time you want to use. When the value of curNonce increased MAX_NONCE_NUM, protocol agreement to stop the operation of the main loop , enter idle state.

MSG_TYPE_NUM constant value is defined as 3, because only three types of messages. These three messages ID 1, 2 and 3, respectively, so the statement typedef int [1, MSG_TYPE_NUM] MsgType; defines the message ID data type MsgType, statements const MsgType Msg1 = 1, Msg2 = 2, Msg3 = 3; definition these three types of constant news Msg1, Msg2 and Msg3.

The intruder intercepts messages stored in accordance with the three different types of arrays. The three arrays for space MAX_S_MSG. When the array space is exhausted , a new alternative intercepted messages last message .

Statement chan msg [MAX_HOST_NUM]; providing a channel msg [ID] for each protocol in accordance with its main ID. Statement Msg curMsg [MAX_HOST_NUM]; with a cache space for storing the message needs to be sent for each channel. Message type Msg contains six items : msg_type is the type of the message ; creator creator store the message ID; sender to store the name of the sender of the message ; receiver store the recipient of the message ; nonce1 and nonce2 store fresh values. When the intruder send fake messages to host identity , creator of the message is the intruder ID, sender is disguised host of ID. nonce1 and nonce2 meaning depends on the message type , as shown in Table 1 . Table of message types and Na, Nb same meaning as described in 6.2 .

Table 1. nonce1 and nonce2 Meaning

Message Type	nonce1	nonce2
1	Na	-
2	Na	Nb
3	-	Nb

Secondly, the protocol model NSPK detection system model for detailed design , detailed design of the first phase of the first based on the general design requirements, to build the system model of cryptographic protocols normal operation . In this case , Network template only has the ability to forward messages. Automata model and template Sender declaration section in Figure 2 and Figure 3, respectively. Automata model and template Receiver declaration section is shown in Figure 5 and Figure 6, respectively. Template Network automaton model shown in Figure 7, the declaration section is empty. Process and system definition shown in Figure 8 .

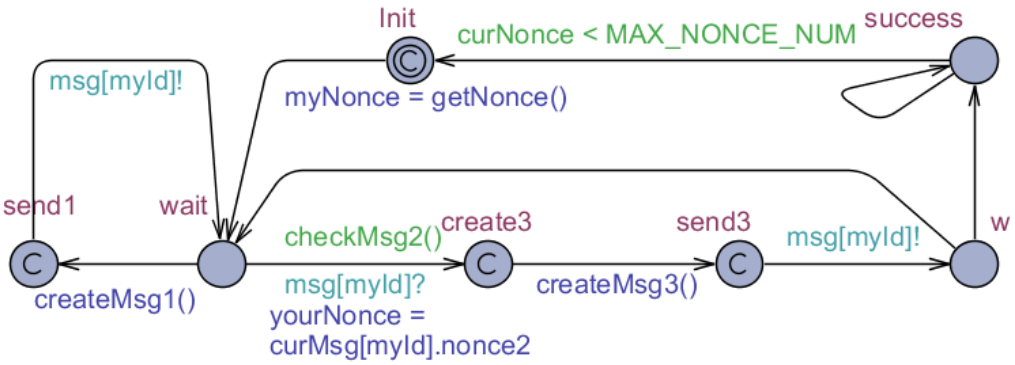


Figure 2. Template Sender Automaton Model

```
Nonce myNonce, yourNonce;
Msg msg1 ={ Msg1, myId, myId, yourId, NullNonce, NullNonce };
Msg msg3 ={ Msg3, myId, myId, yourId, NullNonce, NullNonce };
void createMsg1(){
   if(msg1.nonce1 != myNonce)        msg1.nonce1 = myNonce;
   curMsg[myId] = msg1;
}
bool checkMsg2(){
   if( curMsg[myId].msg_type != Msg2 ) return false;
   if( curMsg[myId].receiver != myId ) return false;
   if( curMsg[myId].sender != yourId ) return false;
   if( curMsg[myId].nonce1 != myNonce) return false;
   return true;
}
void createMsg3(){
   if(msg3.nonce2 != yourNonce)        msg3.nonce2 = yourNonce;
   curMsg[myId] = msg3;
}
```

Sender Template Declaration Section 3 of

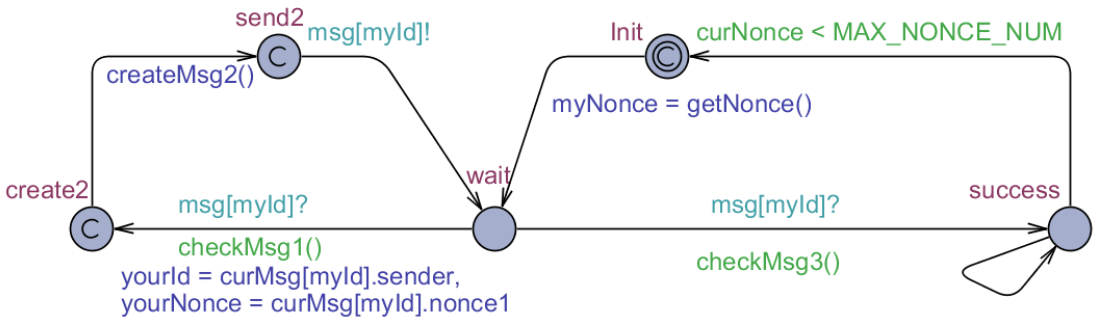
Sender of the initial position of the template is defined as Init Committed, which is to force the generation of fresh value immediately. getNonce function is defined in the global declaration, in accordance with the value of fresh produce fresh value generation method of the overall design. Defined getNonce function is shown in Figure createMsg1 function creates a message, the message is created is stored in curMsg [myId] by msg [myId] channel to send out a message and then wait 2. checkMsg2 function is used to check whether the message received is awaiting news of two, check the project include: Message type is a Msg2, whether the recipients themselves, whether the sender of a predetermined authentication responders, whether the value of the data item is nonce1 own nonce used in the authentication process. If passed checkMsg2 function checks, use the function to create a message createMsg3 3, stored in curMsg [myId] and sends it through msg [myId] channel. Message 1 or Message 3 may not wait for a response message as return to wait position after being sent to resend a message , in this case remains the same fresh values . If the entire certification process is successfully completed, it is determined that a certification process back under Init start position, or idle position under the control of success curNonce value .

```
Nonce getNonce(){
curNonce ++;
return curNonce;
}
```

Figure 4. getNonce Function Definition

Also in order to force the generation of fresh value immediately at the beginning of the certification process, the initial position Init template Receiver is also defined as a Committed. Receiver template first own channel msg [myId] wait for the message, and then the type of call checkMsg1 function checks whether the message is a message. By checking checkMsg1 function after creating a message using createMsg2 function 2, stored in curMsg

[myId] in and msg in the channel [myId] sent, and then wait for the message in the wait position. Message 3, check whether the project received a message waiting checkMsg3 function checks include : The message type is a message 3, the sender is based sub certification process certification originator, recipient whether their own, whether the value of the data item nonce2 fresh value -oriented sub- certification process they use. If the message is received through the inspection checkMsg3 function, enter the location of success. Then , under the control curNonce value back to a certification process to determine the location Init start the next , or the success idle position .

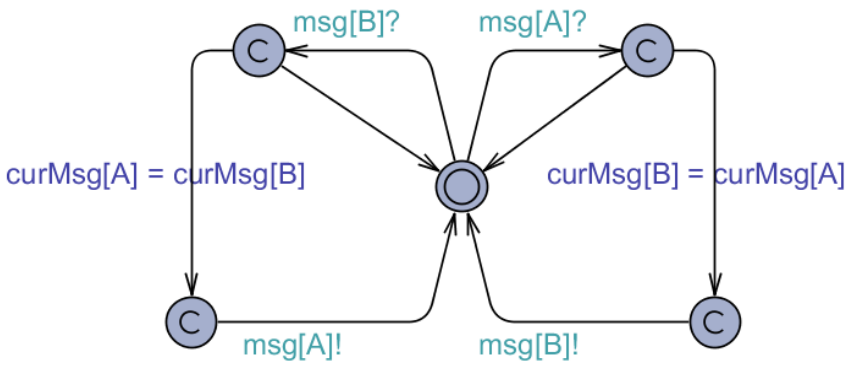


Automata model of Figure 5 template Receiver

```
HostId yourId;
Nonce myNonce ;
Nonce yourNonce ;
Msg msg2 ={ Msg2, myId, myId, NullHost, NullNonce, NullNonce};
bool checkMsg1(){
   if( curMsg[myId].msg_type != Msg1 ) return false;
   if( curMsg[myId].receiver != myId ) return false;
   return true;
}
void createMsg2(){
   if(msg2.receiver != yourId)       msg2.receiver = yourId;
   if(msg2.nonce1 != yourNonce )         msg2.nonce1 = yourNonce;
   if(msg2.nonce2 != myNonce )        msg2.nonce2 = myNonce;
   curMsg[myId] = msg2;
}
bool checkMsg3(){
   if( curMsg[myId].msg_type != Msg3 ) return false;
   if( curMsg[myId].receiver != myId ) return false;
   if( curMsg[myId].sender != yourId ) return false;
   if( curMsg[myId].nonce2 != myNonce) return false;
   return true;
}
```

Declaration Section 6 of the Template Receiver

Network automaton model template is a secure network, and can only receive and forward messages, and messages may be lost. The system has three processes Alice, Bob, and Network.



Automata Model of Figure 7 Template Network

```
Alice = Sender(A,B);
Bob = Receiver(B);

system Alice, Bob, Network;
```

Figure 8 is Defined Processes and Systems

System model detailed design of this phase was to satisfy the following properties:

1) The system does not deadlock , A [] not deadlock;

2) Alice eventually be able to successfully complete the certification process , E <> Alice.success;

3) Bob eventually successfully completed the certification process , E <> Bob.success;

4) Alice and Bob ultimately successful completion of the certification process at the same time , E <> Alice.success && Bob.success;

5) When Alice final completion certificate , Bob also confirmed the identity of Alice , E <> Alice.success && Alice.myNonce == Bob.yourNonce && Alice.yourNonce == Bob.myNonce;

6) When Bob finally completed certification , Alice also confirmed the identity of Alice , E <> Alice.success && Alice.myNonce == Bob.yourNonce && Alice.yourNonce == Bob.myNonce;

From Figure 9 it can be seen using UPPAAL Model detector result is the above properties are satisfied, indicating that the target has reached this stage . For modeling NSPK agreement to meet the design requirements.

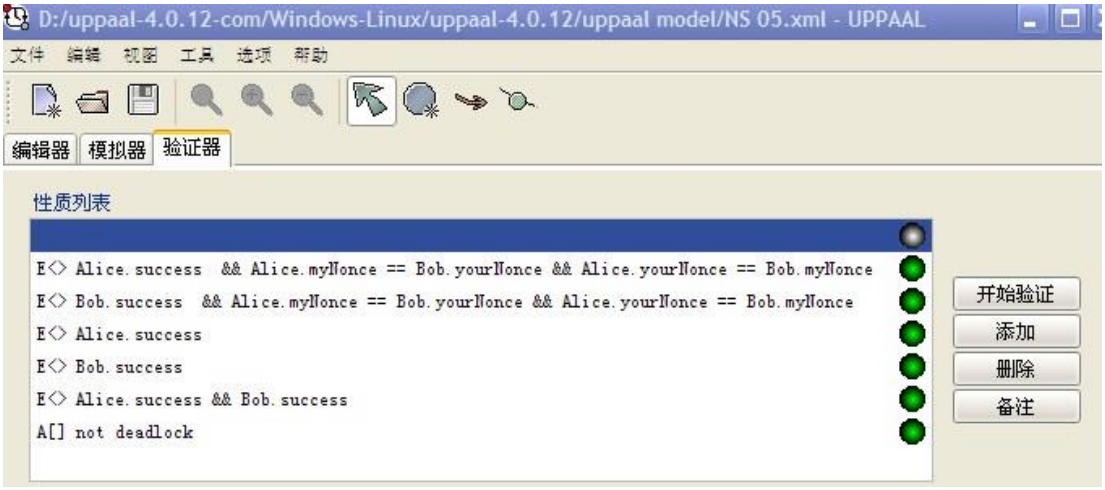


Figure 9. Construction of the Detection Result of the System Model

The second stage of detailed design NSPK protocol security model checking system's mission is to build the required intruder model, so that with the capture, storage, reproduction, camouflage, create functional messages, and the ability to coordinate with the model of cryptographic protocols carried out.

Figure 14 is an intruder model structure . This model features a function to store complete storage intercepted messages. code strore function shown in Figure 10. extract function to use their public key to decrypt the encrypted message to extract and save the message fresh values. extract function shown in Figure 11. repaly function completion message replay function, the code shown in Figure 12. produceMsg function to complete the message and to forge their own identity to create the message function, the code shown in Figure 13 .

```
void store(){
   if( curMsg[from].receiver == myId ) return;
   if(curMsg[from].msg_type == 1){
      if( p1 == 0 || !equals(msg1_S[p1-1],curMsg[from])){
         msg1_S[p1] = curMsg[from];
         if(p1+1 < MAX_S_MSG ) p1++;
     }
   }else if(curMsg[from].msg_type == 2){
      if( p2 == 0 || !equals(msg2_S[p2-1],curMsg[from])){
         msg2_S[p2] = curMsg[from];
         if(p2+1 < MAX_S_MSG ) p2++;
      }
   }else{
      if( p3 == 0 || !equals(msg3_S[p3-1],curMsg[from])){
         msg3_S[p3] = curMsg[from];
         if(p3+1 < MAX_S_MSG ) p3++;
      }
   }
}
```

Figure 10. Store Function Code

```
void extract(){
   if(curMsg[from].receiver == myId){
      if(curMsg[from].msg_type == 1){
         if(curMsg[from].nonce1 != myNonce)    Na = curMsg[from].nonce1;
      }else if(curMsg[from].msg_type == 2){
         if(curMsg[from].nonce1 != myNonce)    Na = curMsg[from].nonce1;
         if(curMsg[from].nonce2 != myNonce)    Nb = curMsg[from].nonce2;
      }else{
         if(curMsg[from].nonce2 != myNonce)    Nb = curMsg[from].nonce2;
      }
   }
}
```

Figure 11. Extract Function Code

```
void replay(int[0,MAX_S_MSG-1] p){
      if(msg_type == 1){
         if( p > p1-1) p = p1-1;
         to = msg1_S[p].receiver;
         curMsg[to] = msg1_S[p];
      }else if(msg_type == 2){
         if( p > p2-1) p = p2-1;
         to = msg2_S[p].receiver;
         curMsg[to] = msg2_S[p];
      }else{
         if( p > p3-1) p = p3-1;
         to = msg3_S[p].receiver;
         curMsg[to] = msg3_S[p];
      }
}
```

Figure 12. Replay Function Code

```
void produceMsg(){
    Msg m;
    m.msg_type = msg_type;
    m.creator = myId;
    m.sender = from;
    m.receiver = to;
    if(msg_type == 1 ){
       m.nonce1 = myNonce;
    }else if(msg_type == 2){
       if( Na != 0 ){
          m.nonce1 = Na;
          m.nonce2 = myNonce;
       }else if( Nb != 0 ){
          m.nonce1 = myNonce;
          m.nonce2 = Nb;
       }else{
          m.nonce1 = myNonce;
          m.nonce2 = myNonce;
       }
    }else{
       m.nonce2 = myNonce;
    }
    curMsg[to] = m;
}
```

Figure 13. produceMsg Function Code

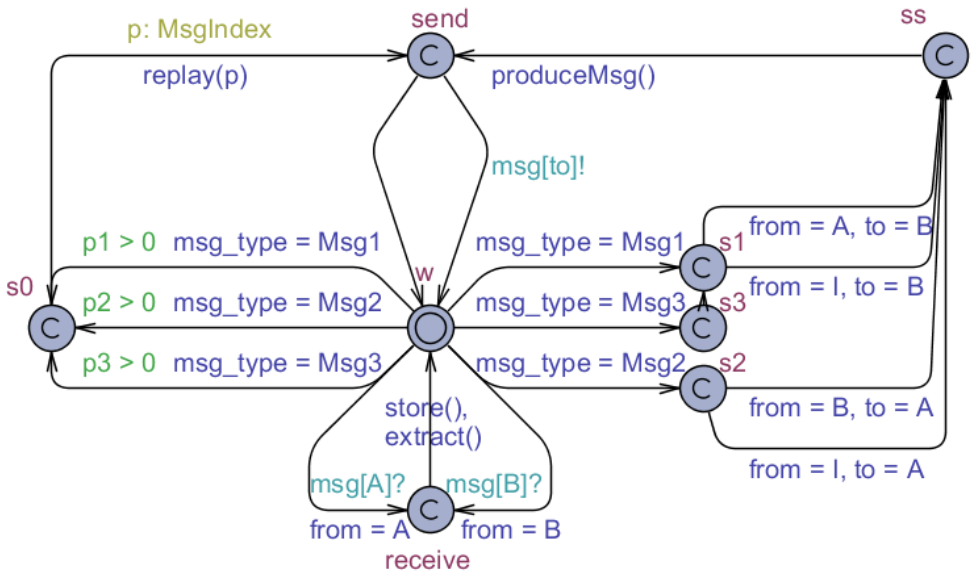


Figure 14. Automata Model Templates Intruder