A Scenario-Based Protocol Checker for Public-Key Authentication Scheme

Takamichi SAITO

Abstract— **Communication security depends on security protocol such like Secure SHell or Secure Socket Layer. One of the important features is authentication. Its correctness is strongly related with the whole of communication security. In this paper, we introduce three types of attack-models that can be actualized as their attack-scenarios, and provide an authentication protocol checker to apply the three types of the attack-scenarios. We also show some problems in security protocols.**

I. Introduction

Security protocol between two parties, SSH[1], [2], SSL/TLS[3], [4], and IPsec are widely utilized for obtaining secure communication. It can ensure *authenticity*, *data integrity* and *confidentiality* over TCP/IP communication. Especially, authenticity provided by authentication scheme is an axis of communication security. Therefore, the security of authentication in security protocol is needed to analyze by utilizing a reliable way.

Many works have been done in these years in this field. NRL Protocol Analyzer [6] is one of the first tools to utilize *reachability* toward the insecure state. The effective result using FDR [7] is known as having found the flaw of the Needham-Schroeder public-key authentication protocol [8]. BAN logic [9] can provide a partial analysis using logic for authentication. More formal and logical methodology is provided by the inductive approach [10]. The recent works [11], [12], using the *strand space*, are focusing on *matching* between the two parties' protocol run. Since a model-checking tool such like SPIN can work effectively in software engineering [13], such formal method with using automatic tool has been applied to security protocol [14].

As the other approach, there are some informal but highly edifying discussions provided by W. Diffie et al. [17], M. Abadi et al. [18] and C. Meadows [19]. They can rather provide the protocol designers some guiding principles. One of the reasons why they can is that those provide some actual and practical examples: authentication protocol itself, their runs and processes of attacks. Especially, the processes can persuade us to design security protocol.

A solution for showing security of a protocol is to specify demonstration of an attack, rather than reasoning in an ideal box. Then, in this paper, for obtaining visible results, we introduce three types of attack-models for authentication scheme to specify the process of an attack. One of them is abstraction of the Man-In-The-Middle (MITM) attack described in many textbooks such like [15], [16], occurred in a *naive* usage of the Diffie-Hellman key-exchange [20] utilized in security protocols. Based on the proposed three attack-models, we also provide the way of making the corresponding scenarios as a concrete process of attack. Since the MITM scenario can be compatible with others, the rests of scenarios are harmonized with the MITM scenario in the proposed system.

Besides the methodology, there is the other important point in practical analysis. It is to abstract the essence from security protocols. For examples, while there are ten messages in SSL Handshake, excluding *ChangeCipherSpec* messages, in *client authentication mode*, the messages related directly with authentication are just four ones. Not as a simple authentication protocol or a cryptographic protocol, security protocol has some functions for obtaining total secure communication. Therefore, when we discuss its security or analysis, we have to decide to select correctly which messages are related with authentication in the protocol. If we misjudge it, the methodology like even a formal method or logic can't be effective to work. Therefore, we propose to implement a scenario-based checker that can deal with an expression of original meaning of security protocols. For expressing security protocols, a simple language is defined in the appendix A, and semantics in the appendix B. Since there are lots of protocols to deal with, they are specified in the Appendix C as reference.

II. System Design

A. Requirements and Design Concepts

Requirements of the proposed system are followings:

1. It can handle the concrete expressions of security protocols.
2. It can provide an attack process based on the attack-model. The process shows to be a series of the attack.
3. It can be expanded to add other new attack-models or checks to analyze additionally.

B. System Design

The implementation consists of the following elements:

1. Server Side
   (a) Web server ··· Apache [21], ver.1.3.29
   (b) Web application ··· PHP, ver.4.3.8 [22]
   (c) Check engine ··· developed by gcc, ver.3.2
2. Client Side
   (a) Web browser ··· IE 6.0.29, Netscape 7.1

The web application and the check engine are implemented in this paper. The detailed features of the check engine are explained in the section III. The web application with a file uploader is for user interface of the engine.

saito@cs.meiji.ac.jp, Meiji University, 1-1-1 Higashimita Tama-ku Kawasaki 214, Japan.

1-4244-0910-1/07/$20.00 ©2007 IEEE

III. Attack-Models and their Scenarios

A. Three Types of Attack-Models

A.1 Fake-Initiator's Attack-Model

In this attack-model, since an initiator A considers an attacker C as a legitimate responder (not B), the initiator A is voluntarily connecting with the attacker's machine. On the other, a responder B considers the connecting as the initiator A. However, in reality, as the attacker C deceives the responder B by masquerading the initiator A, it could finally seize the session key shared between A and B (see Figure 1). This type of attack is defined as *fake-initiator*. This attack-model can be seemed to be abstraction of Lowe's attack [7].

The Needham-Schroeder public-key authentication (NS) protocol is vulnerable to this attack.

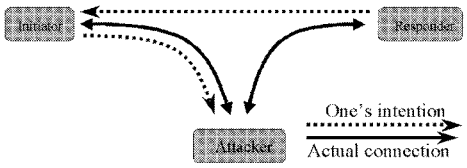

Fig. 1. Image of Fake-Initiator's Attack

For examples, the process of the Lowe's attack [7] is the followings:

$$\begin{array}{lll} (\text{P1}) & A \to \mathcal{I} & : \{A, N_A\}_{P_{\mathcal{I}}} \\ (\text{P1}') & \mathcal{I}(A) \to B & : \{A, N_A\}_{P_B} \\ (\text{P2}) & B \to \mathcal{I}(A) & : \{N_A, N_B\}_{P_A} \\ (\text{P2}') & \mathcal{I} \to A & : \{N_A, N_B\}_{P_A} \\ (\text{P3}) & A \to \mathcal{I} & : \{N_B\}_{P_{\mathcal{I}}} \\ (\text{P3}') & \mathcal{I}(A) \to B & : \{N_B\}_{P_B} \end{array} \tag{1}$$

Where, $\mathcal{I}$ is the attacker itself, but $\mathcal{I}(A)$ is the attacker who masquerades the initiator A.

A.2 Fake-Responder's Attack-Model

In this attack-model, opposite to the fake-initiator's attack-model, since a responder B misidentifies an attacker C as a legitimate initiator (not A), B is connecting with C. On the other, the initiator A considers the connecting as the responder B. However, in reality, as the attacker C deceives the initiator A by masquerading the responder B, it could finally seize the session key shared between A and B (see Figure 2). This type of attack is defined as *fake-responder*. This attack-model is the reverse version of the fake-initiator's attack-model.

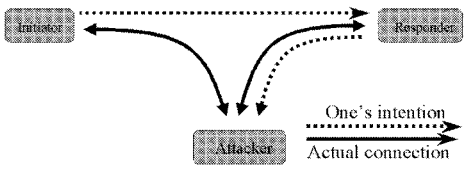

Fig. 2. Image of Fake-Responder's Attack

For examples, now given the following *aho* protocol for this attack:

$$\begin{array}{lll} (\text{P1}) & A \to B & : A \\ (\text{P2}) & B \to A & : \{A, B, N_B\}_{P_A} \\ (\text{P3}) & A \to B & : \{N_A, N_B\}_{P_B} \\ (\text{P4}) & B \to A & : \{H(B, N_A)\}_{N_B} \end{array} \tag{2}$$

The initiator A and the responder B with their public key, i.e., P_A, P_B, try the mutual authentication to share the session key N_B. However, it can't work secure:

$$\begin{array}{lll} (\text{P1}) & A \to \mathcal{I}(B) & : A \\ (\text{P1}') & \mathcal{I} \to B & : \mathcal{I} \\ (\text{P2}) & B \to \mathcal{I} & : \{\mathcal{I}, B, N_B\}_{P_{\mathcal{I}}} \\ (\text{P2}') & \mathcal{I}(B) \to A & : \{A, B, N_B\}_{P_A} \\ (\text{P3}) & A \to \mathcal{I}(B) & : \{N_A, N_B\}_{P_B} \\ (\text{P3}') & \mathcal{I} \to B & : \{N_A, N_B\}_{P_B} \\ (\text{P4}) & B \to \mathcal{I} & : \{H(B, N_A)\}_{N_B} \\ (\text{P4}') & \mathcal{I}(B) \to A & : \{H(B, N_A)\}_{N_B} \end{array} \tag{3}$$

Where, $\mathcal{I}(B)$ is the attacker who masquerades the responder B.

A.3 MITM Attack-Model

This type of attack is considered to be abstraction of the MITM attack. In the model, an initiator A and a responder B are mutually connecting each other, but, in actual connection for an application over TCP/IP, an attacker C can control to obtain all of messages (see Figure 3). It can't prevent over the network without the proper usage of security protocol such as IPsec, SSL and SSH.

As described around security protocols, the *raw* DH protocol is vulnerable to this attack. The example process of this attack is shown in [17]. And checking result shows in the subsection IV-C.

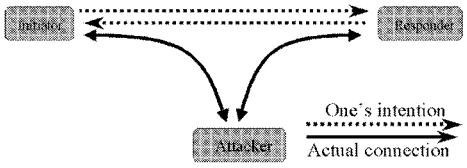

Fig. 3. Image of MITM Attack

B. Scenario of Fake-Initiator's Attack

Instantiation of the attack-model is called as its scenario. The scenario is shown to be a process of an attack, however it is just scenario, therefore it is needed to decide the possibility explained in the subsection III-E after making a scenario.

In this subsection, we explain how to make a scenario of the fake-initiator's attack.

B.1 Preconditions of the Attacker

In this scenario, the attacker holds these following knowledges previously:

1. Symbols, e.g., A, B and C, expressed as parties (the attacker is expressed as C.)
2. Their corresponding public key, e.g., P_A, P_B, and P_C
3. The attacker's secret key: P_C^{-1}
4. The attacker's nonce: $N_x(C)$ where $x \geq 1$ (the attacker can create any number of nonce if it needs.)

B.2 How to Make a Scenario

Now, let the target protocol for checking by the fake-initiator scenario is the followings between the initiator A and the responder B:

$$\begin{array}{ll} \text{(step 1)} & A \to B : msg_1 \\ \text{(step 2)} & B \to A : msg_2 \\ \text{(step 3)} & A \to B : msg_3 \\ \text{(step 4)} & B \to A : msg_4 \\ & \vdots \\ & \vdots \end{array} \quad (4)$$

There are two types of ways to make it scenario: the one is the messages from A to B, the other is the ones from B to A.

B.2.a (1) The Messages from A to B. Let n-th message '$A \to B : msg_n$' where $n = 1, 3, 5, \cdots$. It can be applied by the following procedures, which is called as AtoB-1:

(a) For making a message from A to C, its destination of the transmission is altered. Namely, the symbol B, which means the access with B, is replaced by the symbol C, including in msg_n. The modified message is expressed as msg'_n:

$$\text{(step } n\text{)} \quad A \to C : msg'_n$$

(b) Next, for making the message from $C(A)$ to B, the symbol A of the original is replaced by the symbol $C(A)$, where $C(A)$ means the attacker who masquerading as A. In this case, the message msg_n is not modified:

$$\text{(step } n+1\text{)} \quad C(A) \to B : msg_n$$

B.2.b (2) The Messages from B to A. Let m-th message '$B \to A : msg_m$' where $m = 2, 4, 6, \cdots$. It can be applied by the following procedures, which is called as BtoA-1:

(a) For making a message from B to $C(A)$, its destination of the transmission is altered. Namely, the symbol A is replaced by the symbol $C(A)$. In this case, the message msg_m is not modified as before:

$$\text{(step } m+1\text{)} \quad B \to C(A) : msg_m$$

(b) Next, for making the message from C to A, the symbol B of the original is replaced by the symbol C, including in msg_m. The modified message is expressed as msg'_n:

$$\text{(step } m+2\text{)} \quad C \to A : msg'_m$$

Finally, we can obtain the following process of the attack:

$$\begin{array}{lll} \text{(step 1)} & A \to C & : msg'_n \\ \text{(step 2)} & C(A) \to B & : msg_n \\ \text{(step 3)} & B \to C(A) & : msg_m \\ \text{(step 4)} & C \to A & : msg'_m \\ \text{(step 5)} & A \to C & : msg'_{n+1} \\ & \vdots & \\ & \vdots & \end{array}$$

C. Scenario of Fake-Responder's Attack

In this subsection, we explain about making the scenario of the fake-responder's attack.

C.1 Preconditions of the Attacker

In this scenario, the attacker holds the same knowledges as those described in the subsection III-B.1, too.

C.2 How to Make a Scenario

Now, let the target protocol for the checking is same as the protocol (4) shown in the subsection III-B.2.

Similarly, there are two types of ways to make the scenario: the messages from A to B and the ones from B to A.

C.2.a (1) The Messages from A to B. Let n-th message '$A \to B : msg_n$' where $n = 1, 3, 5, \cdots$. It can be applied by the following procedures, which is called as AtoB-2:

(a) For making the message from A to $C(B)$, its destination of the transmission is altered. Namely, the symbol B is replaced by the symbol $C(B)$. In this case, the message msg_n is not modified:

$$\text{(step } n\text{)} \quad A \to C(B) : msg_n$$

(b) Next, for making the message from C to B, the symbol A of the original is replaced by the symbol C, including in the msg_n. The modified message is expressed as msg'_n:

$$\text{(step } n+1\text{)} \quad C \to B : msg'_n$$

C.2.b (2) The Messages from B to A. Let m-th message '$B \to A : msg_m$' where $m = 2, 4, 6, \cdots$. It can be applied by the following procedures, which is called as BtoA-2:

(a) For making the message from B to C, its destination of the transmission is altered. Namely, the symbol A is replaced by the symbol C, including in msg_m. The modified message is expressed as msg'_m.

$$\text{(step } m+1\text{)} \quad B \to C : msg'_m$$

(b) Next, for making the message from C(B) to A, the symbol B of the original is replaced by the symbol $C(B)$. In this case, the message msg_m is not modified:

$$\text{(step } m+2\text{)} \quad C(B) \to A : msg_m$$

D. Scenario of MITM Attack

In this subsection, we explain how to make the scenario of the MITM attack, it can be just for the key-exchange protocol such like DH. Therefore, it can work with the fake-initiator and the fake responder.

D.1 Preconditions of the Attacker

In this scenario, the attacker holds these following knowledges:

1. Symbols, i.e., A, B and C, expressed as parties
2. The attacker's nonce: $N_x(C)$ where $x \geq 1$ (the attacker can create any number of nonce if it needs.)
3. Random numbers, i.e., w and z, created by the attacker
4. The generator g shared between an initiator A and a responder B

D.2 How to Make a Scenario

Now, let the target protocol for check is same as the protocol (4) in the subsection III-B.2 between an initiator A and a responder B.

There are two types of ways to make the scenario: the messages from A to B and the ones from B to A.

D.2.a (1) The Messages from A to B. Let n-th message '$A \to B : msg_n$' where $n = 1, 3, 5, \cdots$. It can be applied by the following procedures, which is called as AtoB-3:

(a) For making the message from A to $C(B)$, its destination of the transmission is altered. Namely, the symbol B is replaced by the symbol $C(B)$. In this case, the message msg_n is not modified:

$$\text{(step } n\text{)} \quad A \to C(B) : msg_n$$

(b) Next, for making the message from $C(A)$ to B, its source of the transmission is altered. Namely, the symbol A is replaced by the symbol $C(A)$. And, the DH public value is altered by the attacker's one, i.e., g^w, in the message msg_n. The modified message is expressed as msg'_n:

$$(\text{step } n+1) \quad C(A) \rightarrow B : msg'_n$$

D.2.b (2) The Messages from B to A. Let m-th message '$B \rightarrow A : msg_m$' where $m = 2, 4, 6, \cdots$. It can be applied by the following procedures, which is called as BtoA-3:

(a) For making the message from B to $C(A)$, its destination of the transmission is altered. Namely, the symbol A is replaced by the symbol $C(A)$. In this case, the message msg_m is not modified:

$$(\text{step } m+1) \quad B \rightarrow C(A) : msg_m$$

(b) Next, for making the message from $C(B)$ to A, the symbol B is replaced by the symbol $C(B)$. And, the DH public value is altered by the attacker's one, i.e., g^z, in the message msg_m. The modified message is expressed as msg'_m:

$$(\text{step } m+2) \quad C(B) \rightarrow A : msg'_m$$

AtoB-3 can be applied simultaneously with AtoB-1 and AtoB-2, and BtoA-3 can be done with BtoA-1 and BtoA-2. Therefore, in this implementation, the check for the fake-responder and fake-initiator are expanded to collaborate with the MITM attack.

E. Semantics of Attack Scenario

As described before, since the generated scenarios are an only hypothetical process of an attack, it is needed to evaluate its realization as an actual attack. Namely, it is needed to estimate whether the message created by each procedure, e.g., AtoB-1, BtoA-1 and so on, could be provided by the attacker's knowledge in each step, or not. When it evaluates the realization, the attacker can utilize the followings:

1. The pre-knowledge described in the subsection III-B.1, III-C.1 and III-D.1.
2. Knowledges newly obtained until applying the procedure, such like the attacker's received messages or a message made by the received ones.

Only when all messages in each step can be provide by utilizing the above, there is possibility to realize the attack.

Moreover, only in the case, the engine decides whether the attacker holds or creates the shared secret as the session key, or not. If the attacker obtains the shared secret, the engine finally asserts the possibility: `attack success`, which can be seen in the subsection IV-A.1.

From the above discussion, we define the following two concepts around the decision of application of each scenario:

Strong Secure

The attack-scenario can't be realized.

Weak Secure

The shared secret can't be acquired by the attacker.

The former definition could be equivalent to the *secure protocol* defined in the paper of Diffie at el [17]. In this case, the protocol can detect the attack in its run. In other words, in one of steps, there is the message that 'the attacker can't make the message' in the engine's output.

On the other, the latter is not indicated to be vulnerable. In this case, although there is not the assertion, but the attacker can't obtain the shared secret. Namely, the weak secure protocol can't detect the attack in its run. There are examples showed about differences between them in the subsection IV-A.

IV. Examples of Check Results

This section shows some examples with the results. All outputs are printed only in the subsection IV-A.1, but ones in the others are omitted to print. And, for explanation, counting numbers are assigning in left-side.

A. Fake-Initiator's Attack

A.1 Result of NS Protocol

The following results show that NS protocol is vulnerable to the fake-initiator's attack:

```
 1 : authetication protocol checker ver.0.093
 2 : loding file : NS.txt
    (snip)

14 :  --- loading done ------------
15 : check1  :
16 : the attacker's default store :
17 :         { nil , A , B , C , w , z , N:1( C ) }
18 :
19 : stage 1 ----------
20 : step 1:  A => C :  { A , N:1( A ) }_ P:1( C )
21 :
22 : store: { nil , A , B , C , w , z , N:1( C ) , N:1( A ) ,
23 :          { A , N:1( A ) }_ P:1( C ) }
24 :
25 : step 2:  C(A) => B :  { A , N:1( A ) }_ P:1( B )

28 : stage 2 ----------
29 : step 3:  B => C(A) :  { N:1( A ) , N:1( B ) }_ P:1( A )
30 :
31 : store: { nil , A , B , C , w , z , N:1( C ) , N:1( A ) ,
32 :   { A , N:1( A ) }_ P:1( C ) ,
33 :     { N:1( A ) , N:1( B ) }_ P:1( A ) }
34 :
35 : step 4:  C => A :  { N:1( A ) , N:1( B ) }_ P:1( A )

38 : stage 3 ----------
39 : step 5:  A => C :  { N:1( B ) }_ P:1( C )
40 :
41 : store: { nil , A , B , C , w , z , N:1( C ) , N:1( A ) ,
42 :   { A , N:1( A ) }_ P:1( C ) ,
43 :   { N:1( A ) , N:1( B ) }_ P:1( A ) ,
44 :     N:1( B ) , { N:1( B ) }_ P:1( C ) }
45 :
46 : step 6:  C(A) => B :  { N:1( B ) }_ P:1( B )

49 : Attacker can get the shared secret:  N:1( B )
50 : attack success!!
```

L.3 – 14 After parsing the input text loaded from the file, the engine prints its data structure and each parties' knowledges. And the shared secret is specified by the file.

L.17 The printing word '`store`' means to be the attacker's knowledge in this step. Therefore, in this step, it holds the symbols, random numbers and its nonce, which meanings are described in the appendix B.

L.19 In here, the procedure described in the section III-B.2 (1) is applied to the first message from A to B. It is called as the 'stage 1' as in the print.
L.20 This is a result of applying AtoB-1 (a).
L.22, 23 After receiving the message, the attacker gets to hold those knowledges.
L.25 This is a result of applying AtoB-1 (b). In this case, there is no change in the store.
L.28 The procedure described in the section III-B.2 (2) is applied to the second message from B to A. It is called as the 'stage 2' as in the print.
L.29 This is a result of applying BtoA-1 (a).
L.31 – 33 After receiving the message, the attacker holds those knowledges.
L.35 This is a result of applying BtoA-1 (b). In this case, there is no change in the store.
L.38 The procedure AtoB-1(a) is applied to the third message from A to B. It is called as the 'stage 3' as in the print.
L.39 This is a result of applying AtoB-1 (a).
L.41 – 44 After receiving the message, the attacker holds those knowledges. In the 44th line, since there is the shared secret `N:1(B)`, it is evident that the attacker success this attack to obtain it.
L.46 This is a result of applying AtoB-1 (b). In this case, there is no change in the store.

A.2 Example of Weak and Strong Secure

Saito-Hagiya (SH) protocol shown in the appendix C-C is an example of weak secure.

```
 5 : stage 1 ----------
 6 : step 1:  A => C :  { A , N:1( A ) }_ P:1( C )
 7 :
 8 : step 2:  C(A) => B :  { A , N:1( A ) }_ P:1( B )
 9 :
10 : stage 2 ----------
11 : step 3:  B => C(A) :  { N:1( A ) , N:1( B ) }_ P:1( A )
12 :
13 : step 4:  C => A :  { N:1( A ) , N:1( B ) }_ P:1( A )
14 :
15 : stage 3 ----------
16 : step 5:  A => C :  { nil }_ i:1( N:1( B ) )
17 :
18 : step 6:  C(A) => B :  { nil }_ i:1( N:1( B ) )
19 :
20 : Attacker can't get the shared secret:  N:1( B )
21 : attack fail!
```

In this result, while the protocol run is finished without detecting attack, an attacker can't obtain the shared secret.

Next result is a case of strong secure. Hagiya-Saito (HS) protocol[23], which is a revised version of the SH protocol, shown in the appendix C-B.:

```
 3 : stage 1 ----------
 4 : step 1:  A => C :  { A , N:1( A ) }_ P:1( C )
 5 :
 6 : step 2:  C(A) => B :  { A , N:1( A ) }_ P:1( B )
 7 :
 8 : stage 2 ----------
 9 : step 3:  B => C(A) :  { N:1( A ) , N:1( B ) }_ P:1( A )
10 :
11 : step 4:  C => A :  { N:1( A ) , N:1( B ) }_ P:1( A )
12 :
13 : stage 3 ----------
14 : step 5:  A => C :  { C }_ i:1( N:1( B ) )
15 :
16 : step 6:  C(A) => B :  { B }_ i:1( N:1( B ) )
17 :
18 : Attacker can't make this massage.
19 : attack fail!
```

Since the final message in the step 6 can't be created by an attacker, the attack can be detected.

B. Fake-Responder's Attack

The following result shows that NS protocol is not vulnerable to the fake-responder attack. In here, the most messages except its essence are omitted:

```
11 : stage 1 ----------
12 : step 1:  A => C(B) :  { A , N:1( A ) }_ P:1( B )
13 :
14 : store: { nil , A , B , C , w , z , N:1( C ) ,
15 :   { A , N:1( A ) }_ P:1( B ) }
16 :
17 : step 2:  C => B :  { C , N:1( A ) }_ P:1( B )
18 :
19 : Attacker can't make this massage.

   (snip)
```

L.12 This is a result of applying AtoB-2 (a).
L.14 In here, the attacker can't make the message.

C. MITM Attack

The following result shows that DH protocol is vulnerable to the MITM attack, where it is omitted description:

```
stage 1 ----------
step 1:  A => C(B) :  g:1( x )
step 2:  C(A) => B :  g:1( w )

stage 2 ----------
step 3:  B => C(A) :  g:1( y )
step 4:  C(B) => A :  g:1( z )

stage 3 ----------
step 5:  A => C(B) :  { N:1( A ) }_ g:1( x , z )
step 6:  C(A) => B :  { N:1( A ) }_ g:1( w , y )

stage 4 ----------
 step 7:  B => C(A) :  { N:1( A ) , N:1( B ) }_ g:1( w , y )
 step 8:  C(B) => A :  { N:1( A ) , N:1( B ) }_ g:1( x , z )

Attacker can get the shared secret:  N:1( B )
attack success!!
```

In this protocol, the shared secret is utilized only as an encryption key. Then, it needs to check whether the key can be protected or not in the step 7 and 8.

D. All Results

All results including the above ones are shown in this subsection. Other protocols are specified in the appendix C.In the table I, 'I-Attack' is the fake-initiator's attack, 'R-Attack' is the fake-responder's attack and 'M-Attack' is the MITM attack. And, 'X' means to be success of an attack. While 'W' is denoted as the weak secure, 'S' is the strong secure. As some protocols are denoted as abbreviation, the followings are descriptions:

'SSL (RSA:server) + pass' : it is *basic authentication* over *sever authentication* mode using RSA key-exchanging.
'SSL (RSA:client)' : it is *client authentication* mode using RSA key-exchanging.

Protocol	I-Attack	R-Attack	M-Attack
DH	S	S	X
NS	X	S	S
NSL	S	S	S
SH	W	S	S
HS	S	S	S
AHO	S	X	S
STS	S	S	S
SSL (RSA:server) + pass	S	X	S
SSL (RSA:client)	S	S	S
SSL (DH:server) + pass	S	X	S
SSL (DH:client)	S	S	S
SSH1+pass	S	X	S
SSH1+publickey	S	S	S
SSH2+pass	S	X	S
SSH2+publickey	S	S	S

TABLE I
ALL RESULTS OF CHECK

'SSL (DH:server) + pass' : it is *basic authentication* over *sever authentication* mode using DH key-exchanging.
'SSL (DH:client)' : it is *client authentication* mode using DH key-exchanging.
'SSH1+pass' : it is using password authentication of SSH1.
'SSH1+publickey' : it is using public-key authentication of SSH1.
'SSH2+pass' : it is using password authentication of SSH2.
'SSH2+publickey' : it is using public-key authentication of SSH2.

V. CONCLUSION AND FUTURE WORK

In this paper, we introduce three types of attack-models and their scenarios, and defined the concept of 'strong secure' and 'weak secure' of an authentication protocol. Based on the attack-models, we also provide to implement the checker for analyzing the authentication scheme in security protocol. Moreover, we showed that, by using the proposed scenario-based authentication checker, the fake-responder's attack is found in the SSH password-based authentication and the SSL password-based authentication.

As future work, we need some graphical user interface to manipulate this checker. And more samples can be helpful for protocol designers.

REFERENCES

[1] D. Barrett, and R. Silverman: SSH, The Secure Shell, O'REILLY, 2001, ISBN:0-596-00011-1.
[2] T. Ylonen: The SSH (Secure Shell) Remote Login Protocol, Internet Draft, Network Working Group, November 1995.
[3] A. Freier, P. Kocher, and P. Kaltorn: SSL V3.0 Specification, IETF Task Force, March 1996. `http://home.netscape.com/eng/ssl3/s-SPEC`
[4] T. Dierks, C. Allen: The TLS Protocol Version 1.0, 1999, RFC 2246, `http://www.ietf.org/rfc/rfc2246.txt`
[5] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen: SSH Transport Layer Protocol, `draft-ietf-secsh-transport-11.txt`, Internet Draft, Network Working Group, November 2001/
[6] C. Meadows: The NRL Protocol Analyzer: An overview, Journal of Logic Programming, 26(2):113–131, 1996.
[7] G. Lowe: Breaking and fixing the Needham-Schroeder public-key protocol using FDR, Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science vol.1055, pp.147–166, Springer-Verlag, 1996.
[8] R. Needham, and M. Schroeder: Using Encryption for Authentication in Large Networks of Computers, Communications of the ACM, 21, pp.393-399, 1978.
[9] M. Burrows, M. Abadi, and R. Needham: A Logic of Authentication, ACM Transactions on Computer Systems, Vol. 8, No. 1, Feb 1990, pp. 18-36.
[10] L. C. Paulson: The inductive approach to verifying cryptographic protocols, Journal of Computer Security, 6:85–128, 1998.
[11] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman: Strand spaces: Why is a Security Protocol Correct? *Proceedings of 1998 IEEE Symposium on Security and Privacy,* 1998, pp.160–171.
[12] D. Song: Athena: a New Efficient Automatic Checker for Security Protocol Analysis, *Proceedings of the 12th IEEE Computer Security Foundations Workshop,* 1999, pp.192–202.
[13] G. J. Holzman: The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley, 2004.
[14] P. Ryan and S. Schneider: Modeling and analysis of security protocols. Addison-Wesley, 2001.
[15] W. Stallings : Cryptography and Network Security —Principles and Practice— (2nd ed.), Prentice Hall, 1999.
[16] E. Rescorla: SSL and TLS: Designing and Building Secure Systems, Addison-Wesley, 2001 ISBN 0-201-61598-3.
[17] W. Diffie, P. C. Van Oorschot, and M.J. Wiener: Authentication and authenticated key exchanges, Designs, Codes, and Cryptography, 2:107–125, 1992.
[18] M. Abadi, and R. Needham: Prudent engineering practice for cryptographic protocols. IEEE Trans. Soft. Eng. 22, 1 (Jan.), 6-15.
[19] C. Meadows: What Makes a Cryptographic Protocol Secure? The Evolution of Requirements Specification in Formal Cryptographic Protocol Analysis, LNCS Vol.2618, Springer, pp.10-21, 2003.
[20] E. Rescorla: Diffie-Hellman Key Agreement Method, 1999, RFC2631, `http://www.ietf.org/rfc/rfc2631.txt`
[21] `http://www.apache.org/`
[22] `http://www.php.net/`
[23] M. Hagiya, R. Takemura, K. Takahashi, and T. Saito, Verification of Authentication Protocols Based on the Binding Relation, LNCS Vol.2609, Springer, pp.299-316, 2003.

APPENDIX

I. SYNTAX OF INPUT LANGUAGE

A. Input File Format

The followings are description about format of an input file.

1. A line started with `//` is a comment.
2. A null line is ignored.

`Proc`, `KeyHolder` and `Line`, explained in A-B, are written in the following order in the input file:

```
'//' 'comment  comment'
Proc
':'
'--'
KeyHolder
':'
'=='
Line
'//' 'EOF'
```

B. Language for Protocol

`Proc`, `KeyHolder` and `Line` are defined by expanded BNF as follows:

```
Symb := 'A' | 'B' | 'C' | 'CA' | 'nil'
      | 'x' | 'y' | 'w' | 'z'
      | 'saA' | 'saB'
      | 'uname' | 'pass'
      | 'ack1' | 'ack2'
```

```
Subject := 'A' | 'B' | 'C' | 'CA'

Seq := '0' | '1' | '2' | '3' | '4'
     | '5' | '6' | '7' | '8' | '9'

Key := 'i:' Seq '(' LineList ')'
     | 'P:' Seq '(' LineList ')'
     | 'S:' Seq '(' LineList ')'
     | 'K:' Seq '(' LineList ')'
     | 'h:' Seq '(' LineList ')'
     | 'g:' Seq '(' LineList ')'

Nonce := 'N:' Seq '(' Symb ')'

Line := Symb
      | Key
      | Nonce
      |  '{' LineList '}'
      | 'H(' LineList ')'
      |  '{' LineList '}_' Key

LineList := Line
          | Line ',' LineList

Proc := Subject '->' Subject ':' Line

KeyHolder := '[' KeyHolderList ']_' Subject

KeyHolderList := Key
               | Key ',' KeyHolder
```

II. Semantics of Input Language

```
A : Initiator
B : Responder
C : Attacker
CA : Certificate Authority as in PKI
x : Initiator's random number
y : Responder's random number
w, z : Attacker's random number
P : Pubic key
S : Private key
K : Shared key
i : Value of argument
g : Generator, e.g., g(x)=g^x, g(x,y)=g^{xy}
N : Nonce, e.g. N:1(A) : Initiator's nonce
uname : User's identifier
pass : password for user authentication
ack1, ack2 : Acknowledgement
nil : nil string
```

III. Samples

In this section, we provide some results of samples. For printing, some spaces are omitted from originals.

A. NSL protocol

This is a revised version of NS protocol [7].

```
A -> B : { A , N:1(A) }_P:1(B)
B -> A : { B , N:1(A) , N:1(B) }_P:1(A)
A -> B : { N:1(B) }_P:1(B)
```

B. HS protocol

```
A -> B : { A , N:1(A) }_P:1(B)
B -> A : { N:1(A) , N:1(B) }_P:1(A)
A -> B : { B }_i:1(N:1(B))
```

C. SH protocol

```
A -> B : { A , N:1(A) }_P:1(B)
B -> A : { N:1(A) , N:1(B) }_P:1(A)
A -> B : { nil }_i:1(N:1(B))
```

D. STS protocol

```
A -> B :  g:1(x)
B -> A : {g:1(y), {{ g:1(x), g:1(y)}_S:1(B)}_g:1(x,y)}
A -> B : { { g:1(x) , g:1(y) }_S:1(A) }_g:1(x,y)
```

E. SSL

In here, `B` is SSL client, `A` is SSL server.

E.1 SSL (Server auth. mode: RSA key-exchange)

Basic Authentication over SSL in server authentication mode is executed. `pass` is a shared secret as password.

E.1.a Protocol.

```
B -> A : { B , saB }
A -> B : { A , saA }
B -> A : { saA , saB , N:1(B) }_P:1(A)
B -> A : {H(N:1(B),H(saA, saB, B, N:1(B)))}_i:1(N:1(B))
A -> B : {H(N:1(B),H(saA, saB, A, N:1(B)))}_i:1(N:1(B))
B -> A : { uname , pass }_i:1( N:1(B) )
```

E.1.b Attack Process.

```
stage 1 ----------
step 1:  B => C :  { B , saB }
step 2:  C(B) => A :  { B , saB }
stage 2 ----------
step 3:  A => C(B) :  { A , saA }
step 4:  C => B :  { C , saA }
stage 3 ----------
step 5:  B => C :  { saA , saB , N:1( B ) }_ P:1( C )
step 6:  C(B) => A :  { saA , saB , N:1( B ) }_ P:1( A )
stage 4 ----------
step 7:  B => C :  { H( N:1( B ) ,
          H( saA , saB , B , N:1( B )))}_ i:1( N:1( B ))
step 8:  C(B) => A :  { H( N:1( B ) ,
          H( saA , saB , B , N:1( B )))}_ i:1( N:1( B ))
stage 5 ----------
step 9:  A => C(B) :  { H( N:1( B ) ,
          H( saA , saB , A , N:1( B )))}_ i:1( N:1( B ))
step 10: C => B :  { H( N:1( B ) ,
          H( saA , saB , C , N:1( B )))}_ i:1( N:1( B ))
stage 6 ----------
step 11:  B => C :  { uname , pass }_ i:1( N:1( B ))
step 12:  C(B) => A :  { uname , pass }_ i:1( N:1( B ))
Attacker can get the shared secret:  pass
attack success!!
```

E.2 SSL (Client auth. mode: RSA key-exchange)

`N:1(B)` is a shared secret as the session key.

```
A -> B : { A , saA }
B -> A : { B , saB }
B -> A : { saA , saB , N:1(B) }_P:1(A)
B -> A : { H( N:1(B), H( saA, saB, A, B,
           N:1(B) )) }_S:1(B) // Attacker can't make this.
B -> A : {H(N:1(B), H(saA, saB, B, N:1(B)))}_i:1(N:1(B))
A -> B : {H(N:1(B), H(saA, saB, A, N:1(B)))}_i:1(N:1(B))
```

E.3 SSL (Server auth. mode: DH key-exchange)

Basic Authentication over SSL handshake in server authentication mode is executed. `pass` is a shared secret.

E.3.a Protocol.

```
B -> A : { B , saB }
A -> B : { A , saA }
A -> B : { g:1(x) , { H( saA , saB , g:1(x) ) }_S:1(A) }
B -> A : g:1(y)
B -> A : {H(g:1(x,y), H(saA, saB, B, g:1(x,y)))}_g:1(x,y)
A -> B : {H(g:1(x,y), H(saA, saB, A, g:1(x,y)))}_g:1(x,y)
B -> A : { uname , pass }_i:1( g:1(x,y) )
```

E.3.b Attack Process.

```
stage 1 ----------
step 1:  B => C :  { B , saB }
step 2:  C(B) => A :  { B , saB }
stage 2 ----------
step 3:  A => C(B) :  { A , saA }
step 4:  C => B :  { C , saA }
stage 3 ----------
step 5:  A => C(B) : {g:1(x), {H(saA, saB, g:1(x))}_ S:1(A)}
step 6:  C => B :  {g:1(w), {H(saA, saB, g:1(w))}_ S:1(C)}
stage 4 ----------
step 7:  B => C :  g:1( y )
step 8:  C(B) => A :  g:1( z )
stage 5 ----------
step 9:  B => C :  { H( g:1( w , y ) ,
         H( saA , saB , B , g:1( w , y )))}_ g:1( w , y )
step 10: C(B) => A :  { H( g:1( x , z ) ,
          H( saA , saB , B , g:1( x , z )))}_ g:1( x , z )
stage 6 ----------
step 11: A => C(B) :  { H( g:1( x , z ) ,
          H( saA , saB , A , g:1( x , z )))}_ g:1( x , z )
step 12: C => B :  { H( g:1( w , y ) ,
          H( saA , saB , C , g:1( w , y )))}_ g:1( w , y )
stage 7 ----------
step 13: B => C :  { uname , pass }_ i:1( g:1( w , y ) )
step 14: C(B) => A : { uname , pass }_ i:1( g:1( x , z ))
Attacker can get the shared secret:  pass
attack success!!
```

E.4 SSL (Client auth. mode: DH key-exchange)

`N:1(B)` is a shared secret as the session key.

```
A -> B : { A , saA }
B -> A : { B , saB }
A -> B : { g:1(x) , { H( saA , saB , g:1(x) ) }_P:1(A) }
B -> A : g:1(y)
B -> A : { H( g:1(x,y), H( saA, saB, A, B,
          g:1(x,y) )) }_S:1(B) //Attacker can't make this.
B -> A : { H(g:1(x,y), H(saA, saB, B, g:1(x,y)))}_g:1(x,y)
A -> B : { H(g:1(x,y), H(saA, saB, A, g:1(x,y)))}_g:1(x,y)
```

F. SSH

`B` is SSH client, `A` is SSH server.

F.1 SSH1 (password user authentication)

F.1.a Protocol. `pass` is a shared secret as password.

```
A -> B : { P:1(A) , P:2(A) , saA , x }
B -> A : { saB , y , {{  N:1(B) }_P:1(A)}_P:2(A) }
A -> B : { ack1 }_i:1(N:1(B))
B -> A : { uname }_i:1(N:1(B))
A -> B : { ack1 }_i:1(N:1(B))
B -> A : { pass }_i:1(N:1(B))
A -> B : { ack2 }_i:1(N:1(B))
```

F.1.b Attack Process.

```
stage 1 ----------
step 1:  A => C(B) :  { P:1( A ) , P:2( A ) , saA , x }
step 2:  C => B :  { P:1( C ) , P:2( C ) , saA , w }
stage 2 ----------
step 3:  B => C :  { saB, y, { {N:1(B)}_P:1(C) }_P:2(C) }
step 4: C(B) => A : { saB, z, { {N:1(B)}_P:1(A) }_P:2(A)}
stage 3 ----------
step 5:  A => C(B) :  { ack1 }_ i:1( N:1( B ) )
step 6:  C => B :  { ack1 }_ i:1( N:1( B ) )
stage 4 ----------
step 7:  B => C :  { uname }_ i:1( N:1( B ) )
step 8:  C(B) => A :  { uname }_ i:1( N:1( B ) )
stage 5 ----------
step 9:  A => C(B) :  { ack1 }_ i:1( N:1( B ) )
step 10:  C => B :  { ack1 }_ i:1( N:1( B ) )
stage 6 ----------
step 11:  B => C :  { pass }_ i:1( N:1( B ) )
step 12:  C(B) => A :  { pass }_ i:1( N:1( B ) )
stage 7 ----------
step 13:  A => C(B) :  { ack2 }_ i:1( N:1( B ) )
step 14:  C => B :  { ack2 }_ i:1( N:1( B ) )
Attacker can get the shared secret:  pass
attack success!!
```

F.2 SSH1 (public-key user authentication)

`z` is *modulus*, `N:2(B)` is *challenge*.

```
A -> B : { P:1(A) , P:2(A) , saA , x }
B -> A : { saB , y , {{  N:1(B) }_P:1(A)}_P:2(A) }
A -> B : { ack1 }_i:1(N:1(B))
B -> A : { uname }_i:1(N:1(B))
A -> B : { ack1 }_i:1(N:1(B))
B -> A : { z }_i:1(N:1(B))
A -> B : { { N:2(B) }_P:1(B) }_i:1(N:1(B))
B -> A : { H( A , B , N:1(B) ,
         N:2(B) ) }_i:1(N:1(B)) //Attacker can't make this.
A -> B : { ack1 }_i:1(N:1(B))
```

F.3 SSH2 (password user authentication)

F.3.a Protocol. `pass` is a shared secret as password.

```
A -> B : saA
B -> A : saB
B -> A : g:1(y)
A -> B : { g:1(x) , P:1(A) , { H( saA , saB , P:1(A) ,
           g:1(x) , g:1(y) ,g:1(x,y) )}_S:1(A) }
B -> A : { ack1 }_g:1(x,y)
A -> B : { ack2 }_g:1(x,y)
B -> A : { uname , pass }_g:1(x,y)
A -> B : { ack1 }_g:1(x,y)
```

F.3.b Attack Process.

```
stage 1 ----------
step 1:  A => C(B) :  saA
step 2:  C => B :  saA
stage 2 ----------
step 3:  B => C :  saB
step 4:  C(B) => A :  saB
stage 3 ----------
step 5:  B => C :  g:1( y )
step 6:  C(B) => A :  g:1( z )
stage 4 ----------
step 7: A => C(B) : { g:1(x), P:1(A) , { H(saA , saB, P:1(A),
          g:1( x ) , g:1( z ) , g:1( x , z ) ) }_ S:1( A ) }
step 8:  C => B :  { g:1(w), P:1(C) , { H(saA, saB, P:1(C),
          g:1( w ) , g:1( y ) , g:1( w , y ) ) }_ S:1( C ) }
stage 5 ----------
step 9:  B => C :  { ack1 }_ g:1( w , y )
step 10:  C(B) => A :  { ack1 }_ g:1( x , z )
stage 6 ----------
step 11:  A => C(B) :  { ack2 }_ g:1( x , z )
step 12:  C => B :  { ack2 }_ g:1( w , y )
stage 7 ----------
step 13:  B => C :  { uname , pass }_ g:1( w , y )
step 14:  C(B) => A :  { uname , pass }_ g:1( x , z )
stage 8 ----------
step 15:  A => C(B) :  { ack1 }_ g:1( x , z )
step 16:  C => B :  { ack1 }_ g:1( w , y )
Attacker can get the shared secret:  pass
attack success!!
```

F.4 SSH2 (public-key user authentication)

In this case, `g:1(x,y)` is a shared secret.

```
A -> B : saA
B -> A : saB
B -> A : g:1(y)
A -> B : { g:1(x) , P:1(A) , { H( saA , saB , P:1(A) ,
           g:1(x) , g:1(y) ,g:1(x,y) )}_S:1(A) }
B -> A : { ack1 }_g:1(x,y)
A -> B : { ack2 }_g:1(x,y)
B -> A : { uname , P:1(B) , { uname , P:1(B) , A , B ,
    g:1(x,y) }_S:1(B) }_g:1(x,y) //Attacker can't make this.
A -> B : { ack1 }_g:1(x,y)
```