Research Article

A Model Based Security Testing Method for Protocol Implementation

Yu Long Fu¹ and Xiao Long Xin²

¹ LIUPPA, University of PAU and Academy of Bordeaux, 40000 Mont de Marsan, France

² Department of Mathematics, Northwest University, Xi'an 710069, China

Correspondence should be addressed to Xiao Long Xin; xlxinx@126.com

Received 11 March 2014; Accepted 23 May 2014; Published 6 July 2014

Academic Editor: Fei Yu

Copyright © 2014 Y. L. Fu and X. L. Xin. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The security of protocol implementation is important and hard to be verified. Since the penetration testing is usually based on the experience of the security tester and the specific protocol specifications, a formal and automatic verification method is always required. In this paper, we propose an extended model of IOLTS to describe the legal roles and intruders of security protocol implementations, and then combine them together to generate the suitable test cases to verify the security of protocol implementation.

1. Introduction

Since the former employee of CIA, Edward J. Snowden, reveals the global surveillance of National Security Agency (NSA) [1], the suspicions over the existing security mechanisms and products spread over the world. As a consequence, the cryptographic scientists may need to reverify the popular cryptography methods and protocol designs (RSA, DES, etc.). The documents revealed from Snowden also imply the back-doors may exist in some popular network devices/services, such as the productions from CISCO, IBM, Microsoft, and Google, which recalls the importance of the security verification over the network productions (including hardware and software if the interactions between the possible intruders and the protocol implementations will not result in any leakage of privacy, the protocol implements are considered as secure). This is because, even when the security protocols are designed perfectly, the back-doors may be also proposed during the implementing of the productions and may make the privacies leak to some potential attackers. In this paper, we are concerned with this important security problem and propose the model and algorithms to verify the security of protocol implementations automatically.

Security protocols are communication protocols, which guarantee the securities (authentication or confidentiality) by

the defined rules and cryptography methods [2]. A protocol implementation is a solid solution of the corresponding protocol specification, which can be one or a part of software/hardware, and is supposed to work as the designs of the protocol. The implementations of security protocol are usually separated into different network devices and are in charge of the security functions to guarantee the communication security. As the importance to those implementations, those declared security functions should be well verified. However, the classic verification method of protocol implementation, the *protocol testing* [3] methods, is not suitable for security verifications. Although several methods have been proposed in the literature [4–6], the problem of security verification over protocol implementations is still an open problem.

Related Works. Recently, the problems of model based security verifications on protocol implementations have attracted more and more attention of academy. Based on the developing security reasoning theorems (PCL [7], Pi-Calculus [8], and HLPSP [9]) and the corresponding automatic model checking approaches (the methods described in [10–12]), the authors of [4] came up with an approach to verify the security of protocol implementation. They use model checking methods to automatically generate the test cases, which contain the possible security flaws of the protocol,

and then execute those test cases on the implementations to verify their security. Although the presented approach fills the gaps between “security protocol analysis” and “protocol implementation verification” to some degree, several problems are still noticed. First of all, the presented model checker is based on ASLan, which is a specific protocol description language, and makes the translation to other model checkers become difficult. Secondly, the generated test cases only contain the counterexamples related to the detected security flaws; other possible messages sequences which can also cause the insecurity (e.g., the message sequences contain some wrong messages formats) are not included. In [5], the authors proposed a mutation based test generation method for security protocol implementation verification. They propose several mutation rules over the conformed messages sequences of the protocol specification to construct the possible counterexamples. Although the concepts of mutations and fuzzy sets [13] are quite useful to simulate the artificial intelligence of computer network, it is difficult to decide the completeness of the proposed mutator.

Paper Contribution. In this paper, by investigating the features of security protocols and the limitations of the existing protocol testing methods, we propose a security extension of Input/Output Transition System to model and test the protocol implementation. The proposed model inherits the clarity of finite automata and can describe the security properties and most of the protocol behaviors with the definition of transition. We also consider the possible network intruder as one part of the system and propose a model to simulate the attack actions. Then, by the proposed models, the complex security protocol actions can be presented in the way of sequence transitions of the reachable graphic model. The possible test cases can be generated through the model with the proposed algorithm.

The following sections are organized as follows. In Section 2, we briefly introduce the used protocol testing method IOLTS and some testing theories over it. In Section 3, from the requirements of security protocol, we analyze the limitations of the existed IOLTS model and present an extended model of IOLTS under some required assumptions. Then, in Section 4, those modeled protocol components are combined as one system and an intruder model is proposed to simulate the malicious attack actions. The test approach and test case generation algorithms are also presented in this section. And finally, in Section 5, we conclude this paper and introduce some future works.

2. Protocol Testing Theory

Protocol testing represents a set of automata based modeling and testing methods to verify the correctness of the protocol implementations, which has been proposed and widely accepted in the industries since Gonenc [14] found the distinguishing sequences in 1970 [3]. Through the serious preset inputs, the unobservable system status and some internal actions can be deduced by checking the corresponding outputs; then the protocol implementations can be verified by the test execution. This verification method has been used to

solve many problems, such as conformance [15], robustness [16], web service security [17], and cloud computing security [18].

There are several models existing in protocol testing methods. But in this paper, we only consider the model of IOLTS. An IOLTS takes the system behavior as labels and serves as a semantic model for various formal specification languages [19]. Normally a finite IOLTS system can be described as a 4-tuple array.

Definition 1 (labeled transition system). A labeled transition system is a 4-tuple array $\langle S, L, T, s_0 \rangle$ where

- (i) S is a finite set of states;
- (ii) L is a finite set of labels, which contains two subsets: input labels L_I and output labels L_U ;
- (iii) T is the transition relation, where $T \subseteq S \times (L \cup \{\tau\}) \times S$;
- (iv) s_0 is the initial state.

If $\exists a \in L_I$, then we write $?a$ to represent that a is an input label; if $\exists b \in L_U$, then we write $!b$ to identify that b is an output label; we have $L_I \cap L_U = \emptyset$, $L_I \cup L_U = L$. If the input and output labels are not identified, the model is labeled transition system (LTS). τ denotes the internal and unobservable (to the tester) actions of the system. A trace means a finite sequence of the observable actions; they are going to be used as test sequences to verify the system. If $\sigma \in L^*$ ($L^* = L \cup \{\tau\}$), then $|\sigma|$ denotes the length of trace of σ . If $q \in S$, then $Out(q)$ denotes all the output labels from q , $In(q)$ denotes all the input labels to q , and $Out(S, \sigma)$ denotes the output of S after σ . We denote the class of all IOLTS over L_I and L_U by $IOLTS(L_I, L_U)$ or $LTS(L_I \cup L_U)$.

Definition 2. Let $P = \langle S, L, T, s_0 \rangle$ be a labeled transition system, s and $s' \in S$, and let $\mu_i \in L \cup \{\tau\}$, $a_i \in L$, and $\sigma \in L^*$; then we have the following notations:

$$\begin{aligned}
 s &\xrightarrow{\mu} s' =_{def} \exists t \in T, t = (s, \mu, s'), \\
 &\quad t.pre = s, \quad t.nex = s', \quad t.lab = \mu \\
 s &\xrightarrow{\mu_1 \dots \mu_n} s' =_{def} \exists s_0, \dots, s_n : \\
 &\quad s = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \dots \xrightarrow{\mu_n} s_n = s' \\
 s &\xrightarrow{\mu_1 \dots \mu_n} =_{def} \exists s' : s \xrightarrow{\mu_1 \dots \mu_n} s' \\
 s &\xrightarrow{\epsilon} s' =_{def} s = s' \text{ or } s \xrightarrow{\tau \dots \tau} s' \\
 s &\xRightarrow{a} s' =_{def} \exists s_1, s_2 : s \xrightarrow{\epsilon} s_1 \xrightarrow{a} s_2 \xRightarrow{\epsilon} s' \\
 s &\xRightarrow{a_1 \dots a_n} s' =_{def} \exists s_0 \dots s_n : \\
 &\quad s = s_0 \xRightarrow{a_1} s_1 \xRightarrow{a_2} \dots \xRightarrow{a_n} s_n = s' \\
 s &\xRightarrow{\sigma} =_{def} \exists s' : s \xRightarrow{\sigma} s'
 \end{aligned}$$

$$\begin{aligned}
\text{trace}(p) &=_{\text{def}} \left\{ \sigma \in L^* \mid p \xRightarrow{\sigma} \right\} \\
\text{init}(p) &=_{\text{def}} \left\{ a \in L \mid p \xRightarrow{a} \right\}.
\end{aligned} \tag{1}$$

A test case te is a specification for the behaviours of a tester, which exists in an experiment to be carried out with an implementation under test (IUT). Such behaviour, like other behaviours, can be specified by an IOLTS [20]. Then the test case t can be described as an IOLTS over some specific action set L_I and L_U and with a test verdict ($pass$ or $fail$) in the end ($te : i \rightarrow \{fail, pass\}, i \in \text{IOLTS}(L_I, L_U)$). In other words, a test case is a set of transition sequence end with test verdict. Generally, the considered actions of the test cases must relate to the specific protocol specification.

3. Transition Models on Security Protocols

Although the IOLTS based testing methods have been proved to be useful in verifying the protocol implementations, it cannot be used to verify the correctness of security protocol implementations, because of some important features of security protocol. Comparing with the normal system protocols, security protocols have the following important features which stop the modeling by IOLTS.

- (i) Security properties: security protocols usually need to consider security properties such as nonce and session id to help the security functions to identify the security. Those security properties are usually the contents of the exchanging or received messages and are not considerable in IOLTS system.
- (ii) Security checking functions: security protocols use several predefined security rules (e.g., comparing the received nonce with the holding one) to verify the security, which are called security checking functions here. Those fundamental security functions are hard to model by IOLTS. Someone may say that the internal action τ of IOLTS can present those functions, but considering that the security properties cannot be distinguished, an input of IOLTS will trigger multiply checking functions and it is meaningless to the testing.
- (iii) Multiple roles: security protocols naturally contain multiple roles (at least one initiator and one responder), while the IOLTS is designed for one system component.
- (iv) Intruder: most of the security protocols are designed to avoid the attacks from intruder, which is considered as participation of protocol. Notice that this intruder is not presented in the security protocol specification, and it is impossible to propose a model with IOLTS.

In order to describe the required features, we propose extensions of IOLTS with several considerations. To identify the security properties, we define the security properties as “partial label” of the inputs actions and define “parsing

state” and “combination state” to construct the transition. This transition then is going to trigger the security checking functions. Secondly, we impact the way of IO automata [21], which defines the internal actions τ with different suffix to identify different internal actions to this extended model. The security checking functions which are triggered by the proposed “partial label transition” can be identified. The methods of solving the problems of “multiple roles” and “intruder” are going to be presented in the next section. Then, we give a model of security extended labeled transition system (SE-LTS) model as follows.

Definition 3. (SE-LTS) A security extended labeled transition system is an extension of IOLTS for the components of security protocol; it is a 4-tuple array $\langle s_0, S_{se}, L_{se}, T_{se} \rangle$, where

- (i) s_0 is the initial state;
- (ii) S_{se} is a set of states and $S_{se} = S_n \cup S_p$, where S_p is a set of “parsing states,” which parses the received input action into partial labels and S_n represents the set of other normal states;
- (iii) L_{se} is a finite set of labels and $\{l \in L_{se} \mid l = l_0 \cdot l_1 \cdots l_n\}$, where l_n is one partial label of l and $\cdot$ represents the function of concatenation. We define L_p as a finite set of partial labels of all labels in L_{se} ; then $\forall l_n \in L_p$. The unobservable internal actions are marked as τ_n , which indicates different internal actions by the suffix. The set of internal actions is marked as Θ ;
- (iv) T_{se} is the transition relation, $T_{se} \subseteq \{S_n \times L_{se} \times S_p\} \cup \{S_p \times L_p \times S_n\} \cup \{S_n \times \Theta \times S_c\} \cup \{S_c \times L_{se} \times S_n\}$.

A partial label is one part of system action which is sufficient to trigger an internal action of the system. A system action may contain one or multiple partial labels.

Definition 4. Let $P = \langle s_0, S_{se}, L_{se}, T_{se} \rangle$ be a SE-LTS with $s, s' \in S_{se}$, and let $\mu_i \in L_{se} \cup \Theta$, $a_i \in L_{se}$, and $a_i = \{a_{i0} \cdot a_{i1} \cdots a_{in}\}$; we have the following notations:

$$\begin{aligned}
s &\xrightarrow{\mu} s' =_{\text{def}} \exists t \in T, \\
t &= (s, \mu, s'), \quad t.pre = s, \quad t.nex = s', \quad t.lab = \mu \\
s &\xrightarrow{\mu_1 \cdots \mu_n} s' =_{\text{def}} \exists s_0, \dots, s_n : \\
&\quad s = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} s_n = s' \\
s &\xrightarrow{\mu_1 \cdots \mu_n} =_{\text{def}} \exists s' : s \xrightarrow{\mu_1 \cdots \mu_n} s' \\
s &\xRightarrow{a} s' =_{\text{def}} \exists s_1, s_3, \dots, s_{2n+1} \in S_p \cup S_c, \\
&\quad s_2, s_4, \dots, s_{2n+2} \in S_n : \\
&\quad s \xrightarrow{a} s_1 \xrightarrow{a_0} s_2 \xrightarrow{\tau_0} \cdots s_{2n+1} \xrightarrow{a_n} s_{2n+2} \xrightarrow{\tau_n} s' \\
\text{Parse}(a) &=_{\text{def}} \{a_0, a_1, \dots, a_n\}.
\end{aligned} \tag{2}$$

Example 5. The Needham-Schroeder Public Key (NSPK) protocol [22] is an asymmetric cryptography based authentication protocol, which defines the handshakes between two participants: the initiator i and the responder r . The brief protocol narration can be presented with the three message exchanges as follows:

- Msg 1. $i \longrightarrow r$ (Ask) : $\{n_i, i\}_{pk_r}$
 Msg 2. $r \longrightarrow i$ (Rpl) : $\{n_i, n_r\}_{pk_i}$ (3)
 Msg 3. $i \longrightarrow r$ (Cfm) : $\{n_r\}_{pk_r}$.

The NSPK protocol assumes the nonces n_i and n_r are completely random and only the one who holds the private key can decrypt the cipher encrypted by the public key. In this case, when the unique n_i is sent to r through *Msg1*, although this message may be captured by some intruders, the n_i can be only obtained by r . Similar to the nonce n_r through message *Msg2*, it can be only obtained by i . Then these unique nonces are used to verify the correctness and authentication of the protocol participations.

It is hard to model and analyze the security issues of security protocols by using the general models of automata, because the important security properties are usually hidden inside the exchanging messages. However, with our SE-LTS model, the authentication processes can be modeled as internal actions and the important security properties are identified and modeled as partial labels, which are going to be used to trigger those internal actions of the protocol participations. Figure 1 presents the receiver of NSPK protocol described by SE-LTS. In this example, $S_n = \{s_0, s_2, s_4, s_6, s_8, s_{10}, s_{12}, s_{14}, s_{15}\}$, $S_p = \{s_1, s_3, s_5, s_7, s_9, s_{11}, s_{13}\}$, and $S_c = \{s_7\}$; $L_{se} = \{?Ack, !Rpl, ?Cfm\}$, where $?Ack = \{?sender \cdot ?receiver \cdot ?cipher\}$, $?Cfm = \{?sender \cdot ?receiver \cdot ?cipher\}$; $\Theta = \{\tau_0, \tau_1, \tau_2, \tau_3, \tau_4\}$, where τ_0 means r records the sender id; τ_1 means r verifies the receiver id in *?Ack* message equal to its id; τ_2 means a successful description; τ_3 means the received sender id in *?Cfm* is equal to the recorded in *?Ack*; τ_4 means the received decrypted message is equal to n_r . The mark of “ $\neg$ ” means the conditions inverse.

4. Networked Model and Intruder

4.1. Networked Model for Multiple Components. The security protocol implements usually are installed in distributed network devices. To present the connected feature of the security protocol, we use SE-LTS model to describe each form of participation first and then combine them together as a networked transition system. As the method proposed in [23], a communication medium is needed to glue those components. The normal state $s \in S_n$ of $LTS_{se}(L)$ is defined with two levels:

- (i) higher_level state $s_i.u$ connects to the environment or other states of the same component;
- (ii) lower_level state $s_i.l$ connects to the states of other components.

FIGURE 1: SE-LTS for NSPK receiver.

Then a common medium is considered by such transition, which begins from the lower_level state of one component and ends with the lower_level of initial state of another component. S_i , L_i denote the states and labels in $LTS_{se}(L_i)$, and S_j and L_j denote the state and labels in $LTS_{se}(L_j)$; then if $\exists !l \in L_i$, $\exists s_i \in S_i$, $!l \in Out(s_i)$, and $\exists s_j \in S_j$, $?l \in L_j$, $?l \in In(s_j)$, the transition of the common medium between $IOLTS_i$ and $IOLTS_j$ is presented as $s_i.l \xrightarrow{!l} s_0.l$. We make S_{medium} to denote all the states and T_{medium} to denote all the transitions in the medium.

Definition 6 (networked SE-LTS). The implementation of a security protocol contains a set of SE-LTS systems ($LTS_{SE}(L_n)$; L_n represents the set of labels of the n th components, $n = 0, 1 \dots$), where those systems are connected sequentially by some transitions of the medium. It is also a four-tuple array: $\langle s_{0sp}, S_{sp}, L_{sp}, T_{sp} \rangle$, where

- (i) s_{0sp} is the set of initial states of $LTS_{SE}(L_n)$, $n = 0, 1 \dots$;
- (ii) S_{sp} is a set of states and $S_{sp} = S_{LTS_{SE}(L_0)} \cup S_{LTS_{SE}(L_1)} \cup \dots \cup S_{LTS_{SE}(L_n)} \cup S_{medium}$;
- (iii) L_{sp} is a finite set of labels, $L_{sp} = L_{LTS_{SE}(L_0)} \cup L_{LTS_{SE}(L_1)} \cup \dots \cup L_{LTS_{SE}(L_n)}$;
- (iv) T_{sp} is the transition relation, $T_{sp} = T_{LTS_{SE}(L_0)} \cup T_{LTS_{SE}(L_1)} \cup \dots \cup T_{LTS_{SE}(L_n)} \cup T_{medium}$.

Example 7. The networked model of NSPK protocol can be presented as in Figure 2. The transitions between two communications are the medium transition. The lower_interface and higher_interface represent the same state.

4.2. Intruder of Security Protocol Communication. With the help of networked SE-LTS model (noted as $NLTS_{se}(L)$), the specification of security protocol can be modeled as transition system. The sequential actions designed in security protocols are modeled as transition sequences of the presented

FIGURE 2: Glued security extension graph of NSPK.

reachable graph. Now, we need to consider the following question: “what kind of test cases (transition sequences) can be used to verify the security of the protocol implementations?” To address this, we give a soundness definition of the security of protocol implementations.

Definition 8 (soundness security of protocol implementation). If the interactions between the possible intruders and the protocol implementations will not result any leakage of privacy, the protocol implements are considered as secure.

The privacies are the values of some security properties, such as nonce and private key. Then the problem changes to be “how to calculate the possible transition traces which contain the interactions between intruder and protocol implementations” and “how to distinguish the leakage of privacy.” To address these questions, we need to simulate the intruder model in the proposed networked SE-LTS model.

4.2.1. Testable Intruder Model. An intruder is a powerful agent, which may participate in the protocol executions but with a purpose to attack the system. An intruder usually pretends as a legal role and has the abilities to eavesdrop, insert, or intercept the communicating messages between the legal agents. An intruder usually executes a man-in-the-middle attack. We consider the network intruder with the following assumptions.

- (1) Dolev and Yao intruder assumption: the Dolev and Yao intruder assumption was proposed since 1983 with two main properties [24]:

- (a) the cryptography is assumed to be perfect: a message can only be decrypted by someone who has the proper key (there is no way to crack the scheme);
- (b) the messages are considered to be abstract terms: either the intruder learns all messages inside the encryption (because he has the key) or he learns the encrypted message.

- (2) Tester is powerful: a tester of the system has more permissions; a tester can operate the exchanging message as the proposed intruder assumption.

And a network intruder has the following features:

- (i) no refusal: an intruder can pretend as any legal roles of the security protocol and participate in the protocol execution. It accepts all types of output messages from any legal/illegal participations and gives the corresponding response messages (including attack messages);
- (ii) knowledge learning: an intruder can parse the received messages, analyze them, and update its knowledge. This knowledge may contribute to the generation of an executable attack message;
- (iii) message matching: the output messages of the intruder are not generated randomly; they are going to be accepted by the destined principle. The outputs are generated through the intruder knowledge and will match the format of the input messages of the destination.

FIGURE 3: Intruder model of NSPK.

The intruder model needs to present those listed features and must be compatible with the models of the specifications. According to our assumption, the testable intruder is under control of the tester, so all of its functions (including the security functions) are reachable and observable to the tester. Here we simply use two categories of actions: *forward* and *modify*, which are triggered by the partial labels of the received messages. An intruder model of SE-LTS is defined as follows.

Definition 9 (testable intruder model). A testable intruder model of a specific protocol $\langle s_{0,sp}, S_{sp}, L_{sp}, T_{sp} \rangle$ is a SE-LTS system, which is a 4-tuple array $\langle s_0, S_{int}, L_{int}, T_{int} \rangle$, where $L_{int} \cap L_{sp} = L_{U,sp}$ and $\forall a_i \in L_{int}, \forall a' \in Parse(a_i), \exists s, s' \in S_{int}$,

$$s \xrightarrow{a' \{forward, modify\}} s'. \quad (4)$$

An example of testable intruder model of NSPK protocol is presented in Figure 3.

4.2.2. Combination of the Intruder Model and the Networked SE-LTS. In order to calculate the possible transitions related to the intruder actions, we consider the intruder models that exist inside the communication medium. In this case, all the related actions to the security of protocols specifications are modeled as transitions of one combined transition system and can be presented in one reachable graph. For the example of NSPK protocol, the final result of reachable graph is presented in Figure 4. The used intruder model is presented in Figure 3. We denote this combination model as $CNLTS_{se}(L_{sp})$. Generally, in a security protocol, once the current security session finishes, the security implementations

will always prepare for the next session. So the transition traces of each session of the protocol implementations usually begin and end with initial states. Meanwhile, if the received message is wrong, the implementation will go back to the waiting state and wait for the correct message. Those two kinds of situations should be both considered by the test cases, and we say a *security test case is a self-complete transition trace, which begins from the initial state and ends with the state which is waiting for the inputs from other components*. We say a transition sequence is self-complete if its input actions, which are satisfied by the output of other components, are also included in this transition trace.

4.2.3. Test Cases Generation and Verification. As the model $CNLTS_{se}(L_{sp})$ can be presented with a reachable graph, the problem of calculating the required traces can be solved by searching traces in graph and it is NP-complete. Here we propose a “deep first search algorithm” to calculate the possible traces automatically. Similar to the method we used in [25], the proposed algorithm calculates the traces from the possible end state to the initial state. The pseudocodes are presented in Algorithm 1.

Proof of Algorithm 1. The reason of the correctness of Algorithm 1 can be shown inductively as follows.

- (i) A basic transition machine M only contains one transition t . Then by following the algorithm of “Traceback” $trace_i = \{t\}$; because $t.pre$ is initial state, the Checkglue is called. Because M only contains one transition, $t.stimuli$ is empty. And “check_glue”

FIGURE 4: Intruder contained model, NSPK.

ends; then “Traceback” ends. The transition trace is completely found and recorded.

- (ii) In general, we can assume that the first $n - 1$ transitions of M are traced after $n - 1$ iterations of “Traceback.” Then $trace_i = \{t_0, t_1 \dots t_{n-1}\}$. $\forall t \in trace_i$, $t.stimuli$ is empty.
- (iii) The transition $t = t_n$ is taken as inputs of “Traceback” algorithm. By following the algorithm, t is recorded in $trace_i$, then $trace_i = \{t_0, t_1 \dots t_{n-1}, t_n\}$; $t.pre_state$ is initial state; then “check_glue” is recalled. According to the last step, the first $n - 1$ elements of $trace_i$ do not have stimuli transition. We only need to check whether t_n has stimuli transition. Because t_n is the last transition, if $t_n.stimuli$ is not empty, then another Traceback will be called, and t_n will not be the last transition. So t_n must not have stimuli transition; then $trace_hassti$ is empty. The “check_glue” ends, and “Traceback” ends. $trace_i = \{t_0, t_1 \dots t_{n-1}, t_n\}$. All the transitions of M traveled. $\square$

By using this proposed algorithm through $CNLTs_{se}(L_{sp})$, the transition traces related to the security of protocol implementations (denoted as Tr_{gen}) can be calculated. Then we need to decide the *test verdict* for each transition trace to obtain the security test cases. According to the soundness security of Definition 8, a secure implementation can detect the modifications from the intruder. The test verdict of each transition trace is defined as follows.

Definition 10 (test verdict). If $\sigma \in Tr_{gen}$ is generated from protocol specification S and M is the protocol implementation, then one has the following:

- (i) if σ contains any *modify* action of intruder, the test case $te : \{\sigma \rightarrow fail \mid Out(\sigma, S) = Out(\sigma, M)\}$;
- (ii) if σ does not contain any *modify* actions, the test case $te : \{\sigma \rightarrow pass \mid Out(\sigma, S) = Out(\sigma, M)\}$;
- (iii) otherwise, the test verdict is uncertain.

Then the security test cases to verify the protocol implementations are generated. For example, by using our algorithm, 304 test cases are generated to verify the security of NSPK protocol implementations. In Table 1, we present 5 security test cases as an example. As we know, the NSPK protocol has been proved insecure with the attacks of MITM; one possible attack strategy has been proposed by Gavin Lowe in 1995 and this attack strategy corresponds to Test 4 of our example.

5. Conclusion and Future Works

In this paper, we proposed an extension model of the classic IOLTS for the purpose of security verification of protocol implementation. This extended model defines the nonnegligible security properties with partial labels to identify the security actions. Then by proposing the intruder and the combination model, the security protocol can be modeled. The proposed model can help users to generate the probable transition sequences which contain the interactions between intruder and the protocol implementations. Those generated sequences are the abstract message scenarios to verify the implementations of security protocol. We also presented a corresponding test generation algorithm to automatically generate the test cases. Some test cases of the NSPK protocol are presented also and the MITM attack of NSPK is verified to be concluded in our test cases. In the future work, we

```

Input:
Transitions set  $T_{cnlts}$  of  $CNLTs_{se}(L_s p)$ 
Output:
Transition sequence:  $trace$ 
Begin:
 $ArrayList < transition > trace_{rec};$ 
 $ArrayList < transition > T_{search};$ 
Public static void main(String[] args){
    For each  $t \in T_{cnlts}$ {
        If ( $t.nex \in s_{0_{cnlts}}$ ){
             $trace = Traceback(t, T_{cnlts});$ 
             $Output(trace);$ 
        }
    }
    For each  $t_1 \in T_{cnlts}$ {
        For each  $t_2 \in T_{cnlts}$ {
            If ( $t_1.nex = t_2.nex$ ){
                 $trace = Traceback(t_1, T_{cnlts});$ 
                 $Output(trace);$ 
            }
        }
    }
}
Public transition[] Traceback(transition  $t$ , transition[]  $M$ ){
    If ( $t.pre \notin s_{0_M}$ ){
         $trace_{rec}.add(t);$ 
        For each  $t_{pre} \in Search\_tran(t.pre, M)$ {
             $Traceback(t_{pre}, M);$ 
        }
    }else{
         $Checkglue(t.pre, trace_{rec}, M);$ 
    }
    return  $trace_{rec};$ 
}
Public transition[] Checkglue(state  $s$ ,  $ArrayList < transition > trace_i$ , transition[]  $M$ ){
    For each  $t \in trace_i$ {
        If  $t.lab \in L_{IM}$  and  $t.stimuli \neq$  and  $t.stimuli \notin trace_i$ {
            For each  $t_s \in Search\_tran(s, M)$ {
                If ( $t_s.lab = t.stimuli.lab$ ){
                     $Traceback(t_s, M);$ 
                     $t.stimuli =;$ 
                }else{
                    return  $trace_i.toArray();$ 
                }
            }
        }
    }
}
Public  $ArrayList < transition > Search\_tran$ (state  $s$ , transition[]  $M$ ){
    For each  $t \in T_M$ {
        If ( $t.nex = s$ ){
             $T_{search}.add(t);$ 
        }
    }
    return  $T_{search};$ 
}
End;

```

ALGORITHM 1: Test generation algorithm.

TABLE 1: Some security test cases for NSPK protocol.

Test 1	?Dest, !Ask, ?Ask, ?sid, modify, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Ask, ?Ask, ?sender, τ_{0r} , ?receiver, τ_{1r} , ?cipher, τ_{2r} , !Rpl, ?Rpl, ?sid, forward, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Rpl, ?Rpl, ?sender, fail;
Test 2	?Dest, !Ask, ?Ask, ?sid, forward, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Ask, ?Ask, ?sender, τ_{0r} , ?receiver, τ_{1r} , ?cipher, τ_{2r} , !Rpl, ?Rpl, ?sid, forward, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Rpl, ?Rpl, ?sender, pass;
Test 3	?Dest, !Ask, ?Ask, ?sid, forward, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Ask, ?Ask, ?sender, τ_{0r} , ?receiver, τ_{1r} , ?cipher, τ_{2r} , !Rpl, ?Rpl, ?sid, forward, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Rpl, ?Rpl, ?sender, τ_{0i} , ?receiver, τ_{1i} , ?cipher, τ_{2i} , τ_{3i} , !Cfm, ?sid, forward, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Cfm, ?Cfm, ?sender, τ_{3r} , ?receive, τ_{1r} , ?cipher, τ_{2r} , τ_{4r} , pass;
Test 4	?Dest, !Ask, ?Ask, ?sid, modify, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Ask, ?Ask, ?sender, τ_{0r} , ?receiver, τ_{1r} , ?cipher, τ_{2r} , !Rpl, ?Rpl, ?sid, modify, !sid, ?rid, modify, !rid, ?cipher, modify, !cip, !Rpl, ?Rpl, ?sender, τ_{0i} , ?receiver, τ_{1i} , ?cipher, τ_{2i} , τ_{3i} , !Cfm, ?sid, modify, !sid, ?rid, modify, !rid, ?cipher, modify, !cip, !Cfm, ?Cfm, ?sender, τ_{3r} , ?receive, τ_{1r} , ?cipher, τ_{2r} , τ_{4r} , fail;
Test 5	?Dest, !Ask, ?Ask, ?sid, forward, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Ask, ?Ask, ?sender, τ_{0r} , ?receiver, τ_{1r} , ?cipher, τ_{2r} , !Rpl, ?Rpl, ?sid, forward, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Rpl, ?Rpl, ?sender, τ_{0i} , ?receiver, τ_{1i} , ?cipher, τ_{2i} , τ_{3i} , !Cfm, ?sid, modify, !sid, ?rid, forward, !rid, ?cipher, forward, !cip, !Cfm, ?Cfm, ?sender, τ_{3r} , ?receive, τ_{1r} , ?cipher, τ_{2r} , τ_{4r} , fail;

plan to add the fuzzy sets during the test generation process, which can better simulate the actions of intruder of a network system. Some real network protocols, such as RADIUS and the protocol used in [26], are going to be considered. And the more complex experiments [27] are going to be defined.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

- [1] Wikipedia, "Global surveillance disclosures (2013-present)," 2014, [http://en.wikipedia.org/wiki/Global_surveillance_disclosures_\(2013-present\)](http://en.wikipedia.org/wiki/Global_surveillance_disclosures_(2013-present)).
- [2] Y. L. Fu and O. Koné, "Validation of security protocol implementations from security objectives," *Computers & Security*, vol. 36, pp. 27–39, 2013.
- [3] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines—a survey," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1090–1123, 1996.
- [4] A. Armando, G. Pellegrino, R. Carbone, A. Merlo, and D. Balzarotti, "From model-checking to automated testing of security protocols: bridging the gap," in *Tests and Proofs*, vol. 7305 of *Lecture Notes in Computer Science*, pp. 3–18, 2012.
- [5] F. Dadeau, P. C. Héam, and R. Kheddami, "Mutation-based test generation from security protocols in HLPSP," in *Proceedings of the 4th IEEE International Conference on Software Testing, Verification, and Validation (ICST '11)*, pp. 240–248, Berlin, Germany, March 2011.
- [6] Y. Hsu, G. Q. Shu, and D. Lee, "Model-based approach to security flaw detection of network protocol implementations," in *Proceedings of the 16th IEEE International Conference on Network Protocols (ICNP '08)*, pp. 114–123, Orlando, Fla, USA, October 2008.
- [7] A. Datta, A. Derek, J. C. Mitchell, and A. Roy, "Protocol composition logic (PCL)," *Electronic Notes in Theoretical Computer Science*, vol. 172, pp. 311–358, 2007.
- [8] S. Andova, C. Cremers, K. Gjosteen, S. Mauw, S. F. Mjolsnes, and S. Radomirovic, "A framework for compositional verification of security protocols," *Information and Computation*, vol. 206, no. 2–4, pp. 425–459, 2008.
- [9] AVISPA-Project, "The high level protocol specification language," 2003, <http://www.avispa-project.org/delivs/2.1/d2-1.pdf>.
- [10] S. Salamah, A. Gates, and V. Kreinovich, "Validated templates for specification of complex LTL formulas," *Journal of Systems and Software*, vol. 85, no. 8, pp. 1915–1929, 2012.
- [11] K. W. Kim and J. D. Lee, "On the security of two remote user authentication schemes for telecare medical information systems," *Journal of Medical Systems*, vol. 35, article 17, 2012.
- [12] E. M. Clarke, O. Grumberg Jr., and D. A. Peled, *Model Checking*, The MIT Press, 2000.
- [13] W. Wang and X. L. Xin, "On fuzzy filters of pseudo BL-algebras," *Fuzzy Sets and Systems*, vol. 162, no. 1, pp. 27–38, 2011.
- [14] G. Gonenc, "A method for the design of fault detection experiments," *IEEE Transactions on Computers C*, vol. 19, no. 6, pp. 551–558, 1970.
- [15] W. H. Chen, "An optimization technique for protocol conformance testing based on the wp method," *International Journal of Applied Science and Engineering*, vol. 1, no. 1, pp. 45–54, 2003.
- [16] Y. L. Fu and O. Koné, "Security and robustness by protocol testing," *IEEE System Journal*, no. 99, pp. 1–8, 2012.
- [17] T. Avanesov, Y. Chevalier, M. A. Mekki, and M. Rusinowitch, "Web services verification and prudent implementation," in *Data Privacy Management and Autonomous Spontaneous Security*, vol. 7122 of *Lecture Notes in Computer Science*, pp. 173–189, Springer, Berlin, Germany, 2012.
- [18] A. Gougliadis, I. Mavridis, and V. C. Hu, "Security policy verification for multi-domains in cloud systems," *International Journal of Information Security*, vol. 13, no. 2, pp. 97–111, 2014.
- [19] Y. L. Fu and O. Koné, "Using transition systems to model and verify the implementation of security protocol," in *Proceedings of the 6th International Conference on Security of Information and Networks*, pp. 210–217, Aksaray, Turkey, November 2013.
- [20] J. Tretmans, "Conformance testing with labelled transition systems: implementation relations and test generation," *Computer Networks and ISDN Systems*, vol. 29, no. 1, pp. 49–79, 1996.
- [21] N. A. Lynch and M. R. Tuttle, "An introduction to input/output automata," *CWI Quarterly*, vol. 2, pp. 219–246, 1989.

- [22] G. Lowe, "Breaking and fixing the needham-schroeder public-key protocol using fdr," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 1055 of *Lecture Notes in Computer Science*, pp. 147–166, Springer, Berlin, Germany, 1996.
- [23] Y. L. Fu and O. Koné, "Network security against threatening requests," in *Data Privacy Management and Autonomous Spontaneous Security*, vol. 7122 of *Lecture Notes in Computer Science*, pp. 280–294, Springer, Berlin, Germany, 2012.
- [24] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Information Theory Society*, vol. 29, no. 2, pp. 198–208, 1983.
- [25] Y. L. Fu and O. Koné, "A robustness testing method for network security," in *Global Security, Safety and Sustainability & e-Democracy*, vol. 99 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 38–45, Springer, Berlin, Germany, 2012.
- [26] W. Z. Zhang, Y. Zhang, and T. H. Kim, "Detecting bad information in mobile wireless networks based on the wireless application protocol," *Computing*, pp. 1–20, 2013.
- [27] W. Z. Zhang, X. H. Wang, B. Lu, and T. H. Kim, "Secure encapsulation and publication of biological services in the cloud computing environment," *BioMed Research International*, vol. 2013, Article ID 170580, 8 pages, 2013.