DOI: 10.2298/CSIS120601009W

Modeling and Verifying the Ariadne Protocol Using Process Algebra

Xi Wu[1], Huibiao Zhu[1], Yongxin Zhao[2], Zheng Wang[3], and Si Liu[4]

[1] Shanghai Key Laboratory of Trustworthy Computing
Software Engineering Institute, East China Normal University
3663 Zhongshan Road (North), Shanghai, China, 200062
{xiwu,hbzhu}@sei.ecnu.edu.cn
[2] School of Computing, National University of Singapore, Singapore
zhaoyx@comp.nus.edu.sg
[3] Beijing Institute of Control Engineering, China
wangzheng@sei.ecnu.edu.cn
[4] Department of Computer Science, University of Illinois at Urbana-Champaign
siliu3@illinois.edu

Abstract. Mobile Ad Hoc Networks (MANETs) are formed dynamically by mobile nodes without the support of prior stationary infrastructures. In such networks, routing protocols, particularly secure ones are always the essential parts. Ariadne, an efficient and well-known on-demand secure protocol of MANETs, mainly concerns about how to prevent a malicious node from compromising the route. In this paper, we apply the method of process algebra Communicating Sequential Processes (CSP) to model and reason about the Ariadne protocol, focusing on the process of its route discovery. In our framework, we consider the communication entities as CSP processes, including the initiator, the intermediate nodes and the target. Moreover, we also propose an intruder model allowing the intruder to learn and deduce much information from the protocol and the environment. Note that the modeling approach is also applicable to other protocols, which are based on the on-demand routing protocols and have the route discovery process. Finally, we use PAT, a model checker for CSP, to verify whether the model caters for the specification and the non-trivial secure properties, e.g. nonexistence of fake path. Three case studies are given and the verification results naturally demonstrate that the fake routing attacks may be present in the Ariadne protocol.

Keywords: Formal Verification, CSP, Mobile Ad Hoc Networks, Ariadne.

1. Introduction

Wireless communication technology [12] has become one of the most promising technologies. Mobile Ad Hoc Networks (MANETs) [22,39] consist of groups of wireless mobile devices (laptops, PDAs, sensors, etc.), being completely self-configuring and self-organizing, and are independent of any existing fixed infrastructure. In such networks, nodes can forward the data packets for each

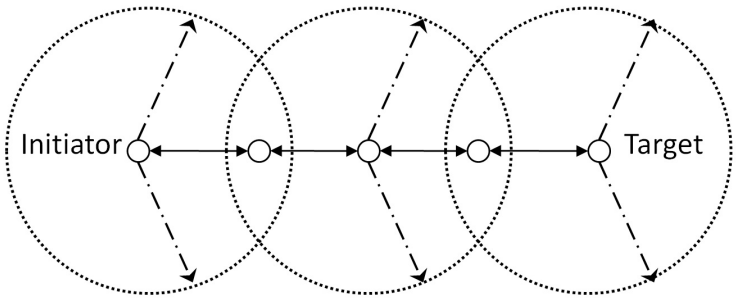

Fig. 1. Communication Model Diagram of Multi-hop Mobile Ad Hoc Network

other through the mutual cooperation. Consequently, even if there is no direct link between two nodes, they also can communicate with each other through the intermediate node multi-hop routing technology, thus widening the range of the data packet transmission. Moreover, nodes can move arbitrarily within, join in, or leave the network dynamically, which makes the whole network quickly and easily set up as needed. Due to these novel features, MANETs have been widely applied in many fields including military, ambient intelligence and emergency contingencies. Figure 1 shows the communication model diagram of multi-hop mobile ad hoc networks.

In such networks, routing protocols [9,10,32,43], particularly secure ones are always the essential factors since they are the major concerns about how to prevent a malicious node from compromising the route. Malicious nodes may cause some typical security issues such as the attacks of denial-of-service and tunneling which redirect the traffic of the networks, the attacks of spoofing that the intruder node may masquerade as the other nodes, and the attack called fabrication of false routing messages. Ariadne [10], as an extension to the dynamic source routing (DSR) protocol [11], proposed by Hu *et al.*, is a new secure on-demand ad hoc network routing protocol for preventing attackers and security vulnerabilities.

Many research efforts have been addressed to analyze and improve the Ariadne protocol. Hu *et al.* evaluated its performance based on simulation [10]. Sivakumar *et al.* proposed some modifications to improve its resiliency [33]. All these works, however, do not investigate the protocol using formal methods and may not take into account the security and correctness. In addition, Lin *et al.* have already found some drawbacks of this protocol, describing them in natural language [13] and Buttyán *et al.* applied a mathematical framework in analyzing the protocol and finding out attacks on it [1,5]. They have done well in analyzing the protocol, if only they had given some verifications. In formal literature, as far as we know, only Pura *et al.* have already modeled the Ariadne protocol using HLPSL and applied AVISPA to validate its security properties [28]. However, they focus more on the use of the tools than the analysis of the protocol itself. Thus, the research for an approach to modeling and verifying the Ariadne

protocol is still challenging. In this paper, we use formal methods to model the protocol and use the process algebra tool PAT to verify whether the achieved model caters for the specification and the non-trivial secure properties.

Lowe *et al.* first applied the method of process algebra Communicating Sequential Processes (CSP) to model and analyze a security protocol, the TMN protocol [19]. CSP is a well-known process algebra in modeling and verifying the reliability, the sequential consistency and the sercurity in concurrent systems and widely used in [18,19,23,29]. Besides, many researchers, such as Zhu and his students, have successfully used CSP to model and analyze the protocols and the web service systems [7,41,42]. Moreover, a lot of automated model checkers for analyzing and understanding systems described by CSP have been produced, such as Process Analysis Toolkit (PAT) [34]. Inspired by Lowe's work, we use CSP to model the Ariadne secure routing protocol. This protocol has two phases: the route discovery and the route maintain. Due to the facts that the route maintain is based on the route discovery and the intrusion tends to occur in the discovery phase, in this paper, we focus on the Ariadne route discovery. We abstract the protocol, that the initiator, the intermediate nodes and the target are described as processes and all of these communication entities share the global clock. It achieves the effect of asymmetric key encryption through clock synchronization and time delay. Besides, we also propose an intruder model in which the intruder can eavesdrop, fake, intercept, learn and deduce the message from the protocol and the environment. Furthermore, by applying PAT [34], we verify the security properties of the Ariadne protocol model and we find that the fake routing attacks may be present in the protocol, which have been pointed out in [1,5]. Finally, we advocate that this suggested framework is also applicable to other protocols, based on the on-demand routing protocols, in which the route discovery process can be modeled as general processes. The main contributions of this paper are listed as follows:

- **Modeling.** A formal model for Ariadne Protocol is given using process algebra CSP. The communication entities of the protocol, including the initiator, the intermediate nodes and the target, are modeled as CSP processes, and we propose an intruder model and produce a CSP description of the specifications.
- **Analysis.** We analyze the whole process of the route discovery of the Ariadne protocol, adding a set of rules into the intruder model. We also give the analysis of the fake path existence in the case studies.
- **Verification.** The formal model is implemented in the model checking tool PAT. The security properties of Ariadne Protocol, i.e., Deadlock Freedom, Message Consistency, Node List Security, Fake Path Nonexistence and End-to-End Nodes Authentication, are verified by PAT. The verification results show that there is a defect in Ariadne Protocol, which may lead to fake routing attacks.

The rest of this paper is organized as follows. We introduce preliminaries about CSP and PAT in Section 2. An overview of the Ariadne secure routing

protocol is presented in Section 3. We formalize the protocol in Section 4 and in Section 5, based on the analysis of traces, we use model checker PAT to implement and verify the achieved model with five properties. In the Section 6, we discuss that the modeling approach presented in this paper is also applicable to other protocols, which are based on the on-demand routing protocols and have the route discovery process. We conclude the paper and present the future directions in Section 7.

2. Preliminaries

2.1. The CSP Method

Process algebra, as a representative of the formal methods, is to use algebraic approaches to study the communications of the concurrent systems. There are three typical calculus systems: Calculus of Communicating Systems (CCS) [24], Communicating Sequential Processes (CSP) and Algebra of Communicating Processes (ACP) [3]. In this subsection, we give a brief introduction to CSP (Communicating Sequential Processes) [8], which was proposed by C. A. R. Hoare in 1978. Nowadays, it has developed and already become one of the more mature process algebra formal method. It specializes in describing the interaction between concurrency systems using mathematical theories. Due to powerful expressive ability, CSP is widely applied in many fields. CSP processes are composed of primitive processes and actions.

The processes in this paper are defined using the following syntax. Here, P and Q represent processes which have alphabets $\alpha(P)$ and $\alpha(Q)$ to denote the set of actions that the processes can perform respectively. a and b stand for the atomic actions and c is the name of channel.

Skip	represents a process which does nothing but terminates successfully.
Stop	denotes that the process is in the state of deadlock and does nothing.
$P; Q$	the process performs P and Q sequentially.
$P \parallel Q$	describes the concurrent between P and Q.
$P[[a \leftarrow b]]$	indicates that a is replaced by b.
$a \rightarrow P$	the process first engages in action a, then the subsequent behavior is like P.
$c?x \rightarrow P$	the process gets a message through the channel c and assigns it to a variable x, then behaves like P.
$c!x \rightarrow P$	the process sends a message x using the channel c, then the behavior is like P.
$a \rightarrow P \Box b \rightarrow Q$	the process behaves like either P or Q and the selection is determined by the environment.
$P \backslash S$	stands for that the process behaves like P except all the actions in set S are concealed.

$P[\|X\|]Q$	the process represents that P and Q perform the concurrent events on the set X of channels.
$P \lhd b \rhd Q$	means if the condition b is true, the behavior is like P, otherwise, like Q.
$CHAOS(x)$	can perform any sequence of events from its alphabet x.

In the verification part, we also apply the trace model of CSP, which is composed of a set of traces, in describing a process. Here, the trace means the events that the process may perform. More details about CSP can be found in [4,8,30,31].

2.2. Process Analysis Toolkit

In this subsection, we give an overview of the verification tool PAT which will be applied in verifying our achieved model of the Ariadne protocol.

PAT (Process Analysis Toolkit) [14,15,16,35] is designed as an extensible and modularized framework for automatic system analysis based on CSP. It supports to specify and verify many different modeling languages and it has been used to model and verify a lot of different systems such as concurrent systems, real-time systems [37], probabilistic systems [38], web service models [36], sensor networks [44,45], and security protocols [20,40]. PAT can be applied in verifying varieties of properties such as deadlock-freeness, divergence-freeness, reachability, LTL properties with fairness assumptions, refinement checking and probabilistic model checking. We list some notations in PAT as follows:

1. $\#define\ N\ 0$ defines a global constant N which has the initial value 0.
2. $var\ msglist[N]$ defines an array named $msglist$ and the size of it is N.
3. $Channel\ c\ 5$ defines a communication channel and the capacity of it is 5.
4. $P = \{x = x + 1\} \rightarrow Skip$ defines an event that can be attached with assignment using which we can update the value of a global variable x.
5. $c!a.b \rightarrow P$ and $c?x.y \rightarrow P$ refer to sending message $a.b$ and receiving message from channel c respectively.

Besides, PAT can also describe the control flow structures, including $if - then - else$ and $while$, etc. More details about this tool can be found in [6,21,34].

3. Overview of the Ariadne Protocol

Ariadne, as an extension to the dynamic source routing (DSR) protocol, is composed of routing discovery and routing maintain. One of its main security goals is to prevent attackers or compromised nodes from tampering with uncompromised routes consisting of uncompromised nodes, and also prevent many types of Denial-of-Service attacks. And it also provides a property that no intermediate node can remove a previous node in the node list in the request or reply, which means that it can prevent a compromised node from removing a node

from the node list arbitrarily [10].

To achieve the security goal as mentioned, the Ariadne protocol applies three authentication mechanisms:

- TESLA protocol [26,27], which is used for route data authentication to certificate the integrity and authenticity of the routing message;
- End-to-End nodes authentication mechanism, which is used to verify the authenticity and freshness of the request and reply using the shared key;
- Per-hop hashing authentication mechanism, which is used to prevent an attacker from removing a node from the node list.

3.1. Notations and Assumptions

In this subsection, we give an overview of the notations and assumptions we will use in our paper. First, we introduce the following notations before describing the protocol:

- S stands for the initiator and D represents for the target;
- A and B are participants, such as the intermediate nodes;
- C denotes an internal node which is captured by the external intruders;
- * stands for all nodes in the whole network;
- K_{SD} and K_{DS}, used to certificate the identities of the communicating nodes, represent the secret MAC keys shared between S and D. The value of K_{SD} is equal to the value of K_{DS};
- $MAC_{K_{SD}}(M)$ denotes the MAC value calculated by the key K_{SD} and the message M.

Besides, during our formalization of the protocol, we also use some assumptions. We naturally inherit all of the assumptions of the Ariadne protocol and the TESLA protocol. In order to facilitate the modeling in the next section, we also list the following assumptions:

1. There is no efficient routing path between S and D in the local table of S.
2. Network is bidirectional [10], i.e., if A can receive the message from B, then B must be able to receive the message from A.
3. Each intermediate node has a TESLA one-way key chain and the keys in the chain are computed through the function $F(x)$. Each node firstly releases one key to all nodes in its broadcast range so that the follow-up nodes can use it to certificate the message. More details about TESLA protocol can be found in [26,27].
4. C, captured by intruders, can get initial information of all the nodes participating the process of route discovery. It can intercept or eavesdrop or fake messages passed between nodes and it can also be used as a normal node to join in the routing process.

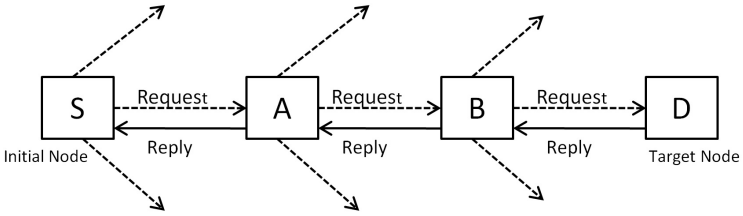

Fig. 2. An Example of the Route Discovery in Ariadne

Note that, security protocol attack models are divided into two types: active attacks and passive attacks. Most of the passive attacks refer to the attacks against the communication privacy, i.e., eavesdropping the data packets between the two communication entities, instead of the attacks against the routing protocols or network functions. On the other hand, active attacks mean that the intruders attack the routing protocols through tampering or faking routing messages to achieve the attack purpose. The sources of the active attacks are also divided into two types: the external intruders and internal intruders [17]. Although they may have the same intrusions, the security threats from the internal intruders are far greater than the ones posed by the external intruders, because the former can get more initial information of the protocol than the latter. Thus, in our paper, we focus more on the security threats brought by the internal intruders.

Using the notations and assumotions as mentioned, we next describe the process of Ariadne routing discovery and its authentication mechanism.

3.2. Description of the Ariadne Routing Discovery and Authentication Mechanism

When there is a packet needed to be sent to the destination, the initiator node first checks its local table. If there is already some path to the target, the initiator will send the packet along the existing path; Otherwise, it will start the route discovery. The process of the Ariadne protocol route discovery is divided into two parts: the initial node broadcasts a routing request to all of its neighbours and the target node sends a routing reply back to the initial node after it gets the request. In order to illustrate the process clearly, we give a simple topology example in Figure 2, where the dotted line means broadcast and the solid line stands for unicast.

Broadcasting the route request. According to Figure 2, S broadcasts a route request to all of its neighbors in the whole network. In Ariadne, a route request contains four fields such as $\langle msg_{req}, hash, node\,list, MAC\,list\rangle$. Here, msg_{req} stands for the route request message and $hash$ value is calculated by a per-hop hash function $H(x)$ except the first element which is computed as $MAC_{k_{SD}}(msg_{req})$.

Through the per-hop hash function $H(x)$, Ariadne protocol ensures that a malicious node cannot remove the intermediate node or modify the order of the node list arbitrarily. $node\ list$ and $MAC\ list$ store the addresses and the MAC values respectively, which are empty initially. msg_{req} also has five fields, $\langle request, initiator, target, id, time\ interval\rangle$. Here we ignore the details of the msg_{req} for simplicity. The process of the broadcasting is shown in Table 1.

Table 1. Broadcasting the Route Request

$S:$	$msg_{req} = (request, S, D, id_{SD}, t_i)$
	$h_S = MAC_{K_{SD}}(msg_{req})$
$S \rightarrow *:$	$\langle msg_{req}, h_S, (), ()\rangle$
$A:$	$h_A = H[A, h_S]$
	$M_A = MAC_{K_{A_{t_i}}}(msg_{req}, h_A, (A), ())$
$A \rightarrow *:$	$\langle msg_{req}, h_A, (A), (M_A)\rangle$
$B:$	$h_B = H[B, h_A]$
	$M_B = MAC_{K_{B_{t_i}}}(msg_{req}, h_B, (A, B), (M_A))$
$B \rightarrow *:$	$\langle msg_{req}, h_B, (A, B), (M_A, M_B)\rangle$

When the intermediate nodes receive the route request, they will first verify the authenticity of the route request message and the effectiveness of the key that the previous node uses. If the TESLA key has already been released, the node will discard this message and send an error message back to the initial node. Otherwise, the current node adds its address, hash value and the MAC value computed by its own TESLA key into the request message and rebroadcasts it.

Unicasting the route reply. After D receives the route request message, it checks the consistency of the $node\ list$ and also tests whether any key the nodes use has already been released within the specified time or not. Only in the case that all the conditions we have mentioned above are satisfied will D accept the route request and send a route reply back to S along the reverse order of the nodes in the $node\ list$.

In Ariadne, a route reply contains three fields, $\langle msg_{rep}, target\ MAC, key\ list\rangle$. Here, $target\ MAC$ stands for the MAC value of the target node and $key\ list$ is empty initially. Besides, msg_{rep} consists of six fields, $\langle reply, target, initiator, time\ interval, node\ list, MAC\ list\rangle$. Table 2 illustrates the process of unicasting the route reply.

Table 2. Unicasting the Route Reply

$D:$	$msg_{rep} = (reply, D, S, t_i, (A, B), (M_A, M_B))$
	$M_D = MAC_{k_{DS}}(msg_{rep})$
$D \rightarrow B:$	$\langle msg_{rep}, M_D, ()\rangle$
$B \rightarrow A:$	$\langle msg_{rep}, M_D, (K_{B_{t_i}})\rangle$
$A \rightarrow S:$	$\langle msg_{rep}, M_D, (K_{B_{t_i}}, K_{A_{t_i}})\rangle$

When an intermediate node receives the route reply, it caches the message until its TESLA key is released. Then the node adds its own key into the $key\ list$ and sends the message to the last node. When S receives the reply, it will check the correctness of each key and each MAC value respectively. Besides, it will also verify the MAC value of D. Only when there is no error in the process of the validation would S accept the route reply and cache the path in its local table. Otherwise, S will discard the reply.

4. Formalizing the Ariadne Protocol

In this section, we use CSP to model the route discovery of the Ariadne protocol. Firstly, we define the sets and channels that we would use below.

- We assume the existence of the set **Initiator** of initiators, the set **Target** of targets.
- The set **Node** stands for the intermediate nodes and **Intruder** represents the internal nodes captured by the malicious nodes.
- The set **SharedKey** contains the keys shared between the initiator and the target.
- The set **Key** involves the TESLA keys that the intermediate nodes use.

In addition, there is another set **MSG**, which stores all the messages passing in the whole route discovery process. We also define four types of the messages as follows:

$$MSG1 =_{df} \{msg_{req}.h_s.S.*|h_s = MAC(K_{SD}, msg_{req}), K_{SD} \in SharedKey, S \in Initiator, * \in Node \cup Target\}$$

$$MSG2 =_{df} \{msg_{req}.h_{X_i}.(..X_j..X_i).(..M_{X_j}..M_{X_i}).X_i.*|* \in Node \cup Target, M_{X_i} = MAC(K_{X_{i_{t_i}}}, msg_{req}, h_{X_i}, (..X_j..X_i), (..M_{X_j}..M_{X_{i-1}})), h_{X_i} = H(X_i, h_{X_{i-1}}), K_{X_{i_{t_i}}} \in Key, X_j, X_i \in Node, i \in N\}$$

$$MSG3 =_{df} \{msg_{rep}.M_D.D.X_i|M_D = MAC(K_{DS}, msg_{rep}), K_{DS} \in SharedKey, i \in N, X_i \in Node, D \in Target\}$$

$$MSG4 =_{df} \{msg_{rep}.M_D.(..K_{X_{j_{t_i}}}..K_{X_{i_{t_i}}}).X_i.S|M_D = MAC(K_{DS}, msg_{rep}), K_{X_{j_{t_i}}}, K_{X_{i_{t_i}}} \in Key, i \in N, X_j, X_i \in Node, S \in Initiator\}$$

$$MSG =_{df} MSG1 \cup MSG2 \cup MSG3 \cup MSG4$$

Here, the specific meaning of each message set can be explained as follows:

1. $MSG1$ represents the set of broadcast messages which are sent by the initiator to the intermediate nodes or the target. Here, h_s stands for the MAC value, which is computed by the key K_{SD}, shared by the initiator and the target, and the request message msg_{req}.

2. The set of $MSG2$ stores the messages that are broadcasted by the intermediate nodes. Any intermediate node receives the request message, it will compute its own hash value and MAC value, then the node will modify the request message using these two values and rebroadcast the request message to the next node.
3. The messages in the set of $MSG3$ stand for the reply messages which are sent by the target to some intermediate node. After the target node receives the request message, it will check the correctness of each value in the message. If all the values are valid, the target node will unicast the reply message according to the reverse order of the intermediate nodes in the node list.
4. The set of $MSG4$ holds the reply messages that are sent back to the initial node.

We use four channels to model the communication in the process of the Ariadne protocol: $Broadcast$, $ComX_iX_j$, $Intercept$, $Fake$.

- $Broadcast$: it is used to broadcast and re-broadcast the message.
- $ComX_iX_j$: it denotes the standard communication between nodes X_i and X_j. Here, nodes include the initiator, the intermediate nodes and the target.
- $Intercept$: the intruder uses it to intercept the information from normal communications.
- $Fake$: it is used by the intruder to send messages which are modified to the normal nodes.

In addition, we also define another two channels: $Session$ and $Fake\_session$, which represent a successful communication and a successful intrusion respectively. The declaration of the channels are as follows:

Channel $Broadcast$, $ComX_jX_i$, $Intercept$, $Fake$: MSG
Channel $Session$, $Fake\_session$: Initiator.Target

Figure 3 illustrates the communications between nodes using channels.

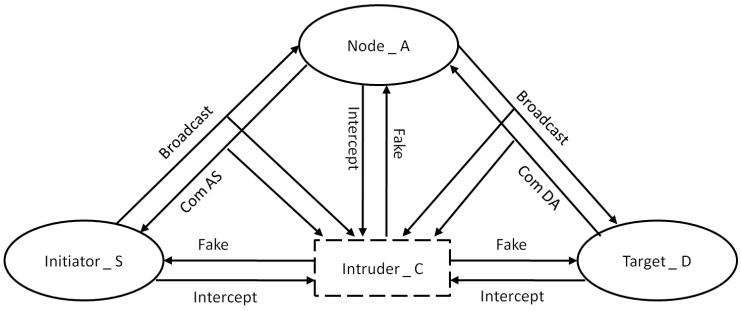

Fig. 3. Communications between Nodes Using Channels

4.1. Clock

The clock is the shared communication entity among the initiator, intermediate nodes and the target. Through clock synchronization and the delaying of releasing the key, the protocol which uses the TESLA broadcasting authentication mechanism achieves the desired effect of asymmetric key encryption.

$Clock(i) =_{df} (tick \rightarrow Clock(i+1))\Box(time?request \rightarrow time!i \rightarrow Clock(i))$
where $i >= 0 \wedge i \in \mathbb{N}$.

Note that, if $Clock$ receives a request from the channel $time$, it will return the current time i.

4.2. Initiator

In the whole process of route discovery, initiator will broadcast the route request and receive the route reply. Besides, it also has to check the validity of each key in the key list and the correctness of the MAC value of the target. Without considering the intruders, we model the behaviors of the initiator node as follows:

$Initiator_1(S, T_0, t_{int}) =_{df} time!request \rightarrow time?T_{S_0} \rightarrow Stop \triangleleft$
$(T_{S_0} > (T_0 + (T+1) * t_{int}) | T_{S_0} < T_0) \triangleright Initiator_2(S, D, K_{SD}, K_{DS})$

The time beginning to send route request is T_{S_0} which must be within the time period from T_0 to T_{T+1}. The time period is divided into $T+1$ small time intervals with a length of t_{int} for each one.

$Initiator_2(S, D, K_{SD}, K_{DS}) =_{df}$
$|||_{X_i \in Node \wedge i \in [1,n] \wedge n \in N}(Broadcast!msg_{req}.MAC(K_{SD}, msg_{req}).S.X_i \rightarrow$
$ComX_iS?msg_{rep}.MAC(K_{DS}, msg_{rep}).\overrightarrow{K_{t_i}}.X_i.S \rightarrow Stop \triangleleft$
$(MAC(K_{SD}, msg_{rep})! = MAC(K_{DS}, msg_{rep})) \triangleright CheckKeyValid_1(\overrightarrow{NL}, \overrightarrow{K_{t_i}})$
$\rightarrow Session.S.D) \rightarrow Initiator_1(S, T_0, t_{int})$

$\overrightarrow{K_{t_i}}$ and $\overrightarrow{NL}$ stand for the key list and the node list respectively. When the initiator receives the reply message, it will check the correctness of the MAC value of the target firstly, through End-to-End nodes authentication mechanism. Then, the process of $CheckKeyValid_1$ is used to check the validation of any key in the key list by the initiator and we give its definition as follows:

$CheckKeyValid_1(\overrightarrow{NL}, \overrightarrow{K_{t_i}}) =_{df} Skip \triangleleft (len(\overrightarrow{K_{t_i}}) == 0) \triangleright$
$(CheckKeyValid_1(tail(\overrightarrow{NL}), tail(reverse(\overrightarrow{K_{t_i}}))) \triangleleft$
$(keymap(head(\overrightarrow{NL}) == F^{i-j}(head(reverse(\overrightarrow{K_{t_i}}))) \triangleright Stop)))$

Here, len shows the length of the list. $head$ and $tail$ return the first element and the remainder of a list respectively. $keymap$ is a function to get the authenticated key at time t_j according to the node id and through $F^{t_i - t_j}(K_{t_i})$ one can get the key K_{t_j} at time t_j. Then, we apply this equation to the received key

value to determine if the computed value matches a previous known authentic key value at time t_j on the key chain. Here, $reverse$ is also a function to reverse the elements in the key list $\overrightarrow{K_{t_i}}$.

Indeed, we must allow the possibility of intruder actions, such as intercepting the request messages and faking the reply messages. We model this situation via the renaming of CSP:

$$
\begin{aligned}
&Initiator(S, T_0, t_{int}) =_{df} Initiator_1(S, T_0, t_{int}) \\
&\quad [[Broadcast!msg_{req}.MAC(K_{SD}, msg_{req}).S.X_i \\
&\qquad \leftarrow Broadcast!msg_{req}.MAC(K_{SD}, msg_{req}).S.X_i, \\
&\quad Broadcast!msg_{req}.MAC(K_{SD}, msg_{req}).S.X_i \\
&\qquad \leftarrow Intercept!msg_{req}.MAC(K_{SD}, msg_{req}).S.X_i, \\
&\quad ComX_iS?msg_{rep}.MAC(K_{DS}, msg_{rep}).\overrightarrow{K_{t_i}}.X_i.S \\
&\qquad \leftarrow ComX_iS?msg_{rep}.MAC(K_{DS}, msg_{rep}).\overrightarrow{K_{t_i}}.X_i.S, \\
&\quad ComX_iS?msg_{rep}.MAC(K_{DS}, msg_{rep}).\overrightarrow{K_{t_i}}.X_i.S \\
&\qquad \leftarrow Fake?msg_{rep}.MAC(K_{DS}, msg_{rep}).\overrightarrow{K_{t_i}}.X_i.S, \\
&\quad Session.S.D \leftarrow Session.S.D, \\
&\quad Session.S.D \leftarrow Fake\_session.S.D]]
\end{aligned}
$$

4.3. Node

The intermediate nodes, in the whole process of the route discovery, have four types of actions: receive the broadcasting request messages from the initiator or the last intermediate node, rebroadcast the modified request messages to the next node, get the unicasting reply messages and pass the reply messages to the intermediate node or the initiator. Before modeling the actions for the intermediate nodes, the definition of the process $Wait(t)$ will be listed below:

$$Wait(t) =_{df} Skip \lhd (t == 0) \rhd (tick \rightarrow Wait(t-1)) \qquad \text{where } t >= 0$$

The process of $Wait(t)$ is used to represent waiting for t time. If the intermediate node receives the reply message but its own TESLA key has not released, it should cache the reply message and wait for a few time until its own key is released. $Wait(t)$ process synchronizes with the global timer and countdowns the time t until it equals to 0. We will give the CSP model ignoring the intruder firstly.

Because the intermediate node uses the TESLA broadcasting authentication protocol for route data authentication, after it receives the request message, it checks whether the key the last node used has already been released or not. If the key has already been released, the node will discard the request, otherwise, it will compute its own hash value and MAC value, and rebroadcast the modified request message to the next nodes. In the model below, X_l stands for the last node, X_i for the current node and X_n for the next node.

$$
\begin{aligned}
&Node_1(S, X_l, X_i, K_{X_{i_{t_i}}}, \delta, T_{S_0}, t_{int}) =_{df} (|||_{X_n \in Node \wedge n \in N \wedge X_n != X_i}(\\
&\quad Broadcast?msg_{req}.\overrightarrow{h_{X_l}}.\overrightarrow{NL_{X_l}}.\overrightarrow{MAC_{X_l}}.X_l.X_i \rightarrow time!request \rightarrow \\
&\quad time?T_{S_1} \rightarrow Checkkey Valid_2(T_{S_0}, T_{S_1}, S, X_l, X_i, \delta, t_{int}) \rightarrow \\
&\quad Broadcast!msg_{req}.\overrightarrow{h_{X_i}}.\overrightarrow{NL_{X_i}}.\overrightarrow{MAC_{X_i}}.X_i.X_n \rightarrow \\
&\quad ComX_nX_i?msg_{rep}.MAC(K_{DS}, msg_{rep}).\overrightarrow{K_{X_{n_{t_i}}}}.X_n.X_i \rightarrow \\
&\quad time!request \rightarrow time?T_{S_2} \rightarrow WaitKeyReleased(T_{S_0}, T_{S_2}, S, X_i, \delta, t_{int}) \rightarrow \\
&\quad ComX_iX_l!msg_{rep}.MAC(K_{DS}, msg_{rep}).\overrightarrow{K_{X_{i_{t_i}}}}.X_i.X_l) \rightarrow \\
&\quad Node_1(X_l, X_i, K_{X_{i_{t_i}}}, \delta, T_{S_0}, t_{int}))
\end{aligned}
$$

$\overrightarrow{h_{X_i}}$ stands for the hash chain constructed using the per-hop hash function $H(x)$, and $\overrightarrow{MAC_{X_i}}$ stands for the current MAC list. $CheckKeyValid_2$ is a process to check the TESLA key the last node used is released or not, and $WaitKeyReleased$ is another process used for node X_i itself to wait until its TESLA key is released at some time. Here, δ represents the number of the time intervals from starting using the key to releasing it. The definitions of the processes $CheckKeyValid_2$ and $WaitKeyReleased$ are given below:

$$
\begin{aligned}
&CheckKeyValid_2(T_{S_0}, T_{S_1}, S, X_l, X_i, \delta, t_{int}) =_{df} \\
&\quad Stop \lhd \\
&\quad ((T_{S_1} + GetTimeDif(X_l, X_i)) >= (T_{S_0} + GetTimeDif(S, X_l) + \delta * t_{int})) \\
&\quad \rhd Skip
\end{aligned}
$$

Here, $GetTimeDif$ is a function to get the time differences between two nodes. According to the TESLA broadcasting authentication protocol, the key the node uses will be released after $\delta * t_{int}$ time from the time it begins to be used.

$$
\begin{aligned}
&WaitKeyReleased(T_{S_0}, T_{S_2}, S, X_i, \delta, t_{int}) =_{df} \\
&\quad Wait(T_{S_0} + GetTimeDif(S, X_i) + \delta * t_{int} - T_{S_2}) \lhd \\
&\quad (T_{S_2} < (T_{S_0} + GetTimeDif(S, X_i) + \delta * t_{int})) \rhd Skip
\end{aligned}
$$

The intermediate node uses the process above to decide whether its key released or not, and it should cache the reply data packet until its own TESLA key is released. Here, we can also use renaming to model the behaviors of the node so as to consider the actions of the intruder as process of $Node$. The intruder may fake or intercept the request and the reply messages, thus we list the detailed model as follows:

$$
\begin{aligned}
&Node(S, X_l, X_i, K_{X_{i_{t_i}}}, \delta, T_{S_0}, t_{int}) =_{df} Node_1(S, X_l, X_i, K_{X_{i_{t_i}}}, \delta, T_{S_0}, t_{int}) \\
&\quad [[Broadcast?msg_{req}.\overrightarrow{h_{X_l}}.\overrightarrow{NL_{X_l}}.\overrightarrow{MAC_{X_l}}.X_l.X_i \\
&\qquad \leftarrow Broadcast?msg_{req}.\overrightarrow{h_{X_l}}.\overrightarrow{NL_{X_l}}.\overrightarrow{MAC_{X_l}}.X_l.X_i, \\
&\quad Broadcast?msg_{req}.\overrightarrow{h_{X_l}}.\overrightarrow{NL_{X_l}}.\overrightarrow{MAC_{X_l}}.X_l.X_i \\
&\qquad \leftarrow Fake?msg_{req}.\overrightarrow{h_{X_l}}.\overrightarrow{NL_{X_l}}.\overrightarrow{MAC_{X_l}}.X_l.X_i, \\
&\quad Broadcast!msg_{req}.\overrightarrow{h_{X_i}}.\overrightarrow{NL_{X_i}}.\overrightarrow{MAC_{X_i}}.X_i.X_n \\
&\qquad \leftarrow Broadcast!msg_{req}.\overrightarrow{h_{X_i}}.\overrightarrow{NL_{X_i}}.\overrightarrow{MAC_{X_i}}.X_i.X_n,
\end{aligned}
$$

$$
\begin{aligned}
&Broadcast!msg_{req}.\overrightarrow{h_{X_i}}.\overrightarrow{NL_{X_i}}.\overrightarrow{MAC_{X_i}}.X_i.X_n\\
&\quad \leftarrow Intercept!msg_{req}.\overrightarrow{h_{X_i}}.\overrightarrow{NL_{X_i}}.\overrightarrow{MAC_{X_i}}.X_i.X_n,\\
&ComX_nX_i?msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{X_{n_{t_i}}}}.X_n.X_i\\
&\quad \leftarrow ComX_nX_i?msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{X_{n_{t_i}}}}.X_n.X_i,\\
&ComX_nX_i?msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{X_{n_{t_i}}}}.X_n.X_i\\
&\quad \leftarrow Fake?msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{X_{n_{t_i}}}}.X_n.X_i,\\
&ComX_iX_l!msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{X_{i_{t_i}}}}.X_i.X_l)\\
&\quad \leftarrow ComX_iX_l!msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{X_{i_{t_i}}}}.X_i.X_l),\\
&ComX_iX_l!msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{X_{i_{t_i}}}}.X_i.X_l)\\
&\quad \leftarrow Intercept!msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{X_{i_{t_i}}}}.X_i.X_l)]]
\end{aligned}
$$

4.4. Target

After the target receives the route request, it checks whether there is any key the nodes use has been released or not through the process $CheckKeyValid_3$ and it also checks the MAC value of the route request using K_{DS}. If there is no error, it will unicast a route reply according to the reverse order of the nodes in the node list. Here, the key list $\overrightarrow{K_{t_i}}$ is an empty list and $\overrightarrow{\Delta t}$ is a list which holds the time difference between every two nodes.

$$
\begin{aligned}
&Target_0(X_l,D,K_{DS},\delta,t_{int},T_{S_0},\overrightarrow{\Delta t}) =_{df}\\
&\quad Broadcast?msg_{req}.\overrightarrow{h_{X_l}}.\overrightarrow{NL_{X_l}}.\overrightarrow{MAC_{X_l}}.X_l.D \rightarrow\\
&\quad time!request \rightarrow time?T_{S_4} \rightarrow\\
&\quad CheckKeyValid_3(\overrightarrow{NL_{X_l}},\delta,t_{int},T_{S_0},\overrightarrow{\Delta t}) \rightarrow\\
&\quad ComDX_l!msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{t_i}}.D.X_l \rightarrow\\
&\quad Session.S.D \rightarrow Target_0(X_l,D,K_{DS},\delta,t_{int},T_{S_0},\overrightarrow{\Delta t})
\end{aligned}
$$

In addition, with respect to the intruder, it can intercept the reply message or fake the request message. We model the target via renaming as follows:

$$
\begin{aligned}
&Target(X_l,D,K_{DS},\delta,t_{int},T_{S_0},\overrightarrow{\Delta t}) =_{df}\\
&\quad Target_0(X_l,D,K_{DS},\delta,t_{int},T_{S_0},\overrightarrow{\Delta t})\\
&\quad [[Broadcast?msg_{req}.\overrightarrow{h_{X_l}}.\overrightarrow{NL_{X_l}}.\overrightarrow{MAC_{X_l}}.X_l.D\\
&\quad \leftarrow Broadcast?msg_{req}.\overrightarrow{h_{X_l}}.\overrightarrow{NL_{X_l}}.\overrightarrow{MAC_{X_l}}.X_l.D,\\
&\quad Broadcast?msg_{req}.\overrightarrow{h_{X_l}}.\overrightarrow{NL_{X_l}}.\overrightarrow{MAC_{X_l}}.X_l.D\\
&\quad \leftarrow Fake?msg_{req}.\overrightarrow{h_{X_l}}.\overrightarrow{NL_{X_l}}.\overrightarrow{MAC_{X_l}}.X_l.D,\\
&\quad ComDX_l!msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{t_i}}.D.X_l\\
&\quad \leftarrow ComDX_l!msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{t_i}}.D.X_l,\\
&\quad ComDX_l!msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{t_i}}.D.X_l\\
&\quad \leftarrow Intercept!msg_{rep}.MAC(K_{DS},msg_{rep}).\overrightarrow{K_{t_i}}.D.X_l,\\
&\quad Session.S.D \leftarrow Session.S.D,
\end{aligned}
$$

$Session.S.D \leftarrow Fake\_session.S.D]]$

4.5. Intruder

In this paper, we also propose an intruder model. Here, the intruder is regarded as a process which can intercept, eavesdrop and fake the message. Through learning, the intruder can get much information about the route discovery. We define a set $Fact$ explaining all the facts it learns.

$$\begin{aligned} Fact =_{df}\ & Initiator \cup Node \cup Target \cup Key \cup SharedKey \cup MSG \\ & \cup\{F(K)|K \in Key\} \\ & \cup\{MAC(K,msg)|K \in SharedKey, msg \in MSG\} \\ & \cup\{H(node,H^{'})|node \in Node\} \\ & \cup\{MAC_i(K,msg,H_i,(..node_j..node_i),(..MAC_j..MAC_{i-1}))| \\ & \quad K \in Key, msg \in MSG, node_j, node_i \in Node\} \end{aligned}$$

The intruder can learn facts from all the sets above. Besides, the intruder node can also deduce some facts from what it has known. We denote $I \mapsto f$ to represent that the intruder deduces new fact f from the known set I.

1. $\{F(K_{X_i})\} \mapsto K_{X_{i-1}}, K_{X_{i-2}}, .., K_{X_0}$
2. $\{MAC(K,msg)\} \mapsto MAC(K,msg)$
3. $\{MAC_i(K,msg,H_i,(..node_j..node_i),(..MAC_j..MAC_{i-1}))\} \mapsto$
   $MAC_i(K,msg,H_i,(..node_j..node_i),(..MAC_j..MAC_{i-1}))$
4. $\{H(node,H^{'})\} \mapsto H(node,H^{'})$
5. $I \mapsto f \wedge I \subseteq \mathbf{I} \Rightarrow \mathbf{I} \mapsto f$

Here, the first deducing rule means that in the TESLA one-way key chain authentication protocol, the intruder can deduce all the previous key values through the one-way key function and any one key. The following three rules stand for the intruder can reason the MAC values and the hash values from the known sets. The last one represents that if the intruder can deduce one new fact f from the known set I, it also can deduce the fact f from the set $\mathbf{I}$, which is bigger than I. As stated in [10], we also assume that the intruder knows the identity of each node so that it can get some information from the fact without deducing. Through the function $info$, the intruder can get various parts of the message. For example:

$$\begin{aligned} & info(msg_{req}.h_S.S.*) = \{msg_{req}, h_S, S, *\} \\ & info(msg_{req}.h_{X_i}.(..X_j..X_i).(..MAC_{X_j}..MAC_{X_i}).X_i.*) = \\ & \quad \{msg_{req}, h_{X_i}, .., X_j, .., X_i, ..MAC_{X_j}, .., MAC_{X_i}, X_i, *\} \\ & info(msg_{rep}.MAC_D.D.X_i) = \{msg_{rep}, MAC_D, D, X_i\} \end{aligned}$$

$$info(msg_{rep}.MAC_D.(..K_{X_{j_{t_i}}}..K_{X_{i_{t_i}}}).X_i.S) =$$
$$\{msg_{rep}, MAC_D, .., K_{X_{j_{t_i}}}, .., K_{X_{i_{t_i}}}, X_i, S\}$$
where $msg_{req}, msg_{rep} \in MSG,\ S \in Initiator,\ * \in Node \cup Target,$
$X_j, X_i \in Node,\ D \in Target,\ K_{X_{j_{t_i}}}, K_{X_{i_{t_i}}} \in Key$

In order to model the behaviors of the intruder, we define another channel $deduce$, through which the intruder can deduce some new facts from the set of the facts it knew. The declaration of channel $deduce$ is:

Channel $deduce : Fact.P\{Fact\}$.

Here, P stands for the power set of the $Fact$. The model of the intruder is as follows:

$Intruder_0(I) =_{df}$
$\Box_{m \in MSG} Broadcast.m \rightarrow Intruder_0(I \cup info(m))$
$\Box$
$\Box_{m \in MSG} ComX_iX_j.m \rightarrow Intruder_0(I \cup info(m))$
$\Box$
$\Box_{m \in MSG} Intercept.m \rightarrow Intruder_0(I \cup info(m))$
$\Box$
$\Box_{m \in MSG} Fake.m \rightarrow Intruder_0(I)$
$\Box$
$\Box_{f \in Fact, f \notin I, I \mapsto f} deduce.f.I \rightarrow Intruder_0(I \cup \{f\})$

We hide the $deduce$ channel because of the internal actions, then we get the model of the intruder:

$$Intruder(I) =_{df} Intruder_0(I) \setminus [|\{deduce\}|]$$

4.6. System and Specification

The whole system can be modeled as the parallel composition of the initiator, the intermediate nodes and the target. All these communication entities share the same clock. First we consider the system without intruder.

$INITIATOR =_{df} Clock(0)[|\{time\}|]Initiator(S, T_0, t_{int})$
$NODE =_{df} Clock(0)[|\{time\}|]Node(S, X_l, X_i, K_{X_{i_{t_i}}}, \delta, T_{S_0}, t_{int})$
$TARGET =_{df} Clock(0)[|\{time\}|]Target(X_l, D, K_{DS}, \delta, t_{int}, T_{S_0}, \overrightarrow{\Delta t})$
$SYSTEM_0 =_{df} INITIATOR[|\{Broadcast, ComX_iS, Session\}|]NODE$
$[|\{Broadcast, ComDX_i, Session\}|]TARGET$

Then, we add the intruder into the whole system.

$$SYSTEM =_{df} SYSTEM_0[|INTRUDER\_ALPH|]INTRUDER(S, D, C, K_{C_{t_i}})$$

$$INTRUDER\_ALPH =_{df} \{|Broadcast, ComX_iX_j, Intercept, Fake, Session, Fake\_session|\}.$$

Ariadne aims to preventing a malicious node from compromising the route. The initiator or the target cannot accept any message modified by the intruder. Here, we define the specification for the security property of the Ariadne protocol as:

$$SPEC =_{df} CHAOS(\sum -\{|Fake\_session.S.D|\}).$$

Note that $\sum$ stands for the set of all the events. As we mentioned in Section 4, the channel $Fake\_session$ represents a successful intrusion between the initiator and the target. In Section 2, $CHAOS(x)$ is explained that the process can perform any sequence of events from its alphabet x. Thus, the whole specification means that the process can perform all the events except the ones on the set of channel $Fake\_session$. Some specific security properties will be listed in the next section.

5. Verification in PAT

In this section, we use PAT to implement our CSP model of the Ariadne protocol. We have already given a brief introduction to PAT in Section 2.

5.1. The Ariadne Protocol in PAT

Here, we implement three cases of our model using PAT. Case I is a basic instance, ignoring the intruder, with a simple topology to show the basic process of route discovery of the Ariadne protocol. Case II and Case III are more complex with two different types of intruders: Case II with the Active-1-1 intruder and Case III with the Active-1-2 intruder. Fake routing attacks may be present in these two cases. Before we implement these three cases in PAT, we first define four other functions in a new $C\sharp$ library as follows:

- $MACValue$ is used only for the initiator and the target to compute their MAC values using the shared key and the message.
- $HashValue$ is used to compute the hash values for the intermediate nodes through function $H(x)$.
- $mediaMACValue$ is used for the intermediate nodes to compute the MAC values using their corresponding TESLA keys.
- $KeyValue$ is a function to compute the node's TESLA key at some time on the TESLA chain.

We also need some significant channels and variables, e.g. $broadcast$ stands for the broadcasting channel, N represents the number of the intermediate nodes in our case, $msgreq$ stands for the request message, $idlist[N+2]$ is a list

that stores the identity for each node, $msgreply[N+1][2]$ is a two-dimensional array that stores the reply messages, $distancelist$ is also a list recording the distance between each two nodes. We give the declarations of them as follows:

```
#define N 4;
channel  broadcast (N+1)*N;
#define msgreq 10;
var msglist[N+1][4];
var idlist[N+2] = [0,1,2,3];
var msgreply[N+1][2];
var distancelist[N+2][N+2]=
                [0,1,2,3,-1,0,1,2,-1,-1,0,1,-1,-1,-1,0];
```

Case I: We implement the basic process of route discovery without intruder. We assume that there are only four nodes such as S, A, B and D which have the same topology as Figure 2, which we have already mentioned in Section 3.

Here, we give the relevant codes in PAT to show how the initiator node sends and receives messages. The initiator S broadcasts the request message to all of its neighbors and it also receives the reply message and does some appropriate checks such as checking the time validation which is interpreted by the process $CheckInitTime(clock)$. We give the PAT code of the processes $SendInits$, $SendInit$ and $SendInit2$ as follows:

```
SendInits(sender, content) = ||| receiver:{1..N+1} @
         (SendInit(sender, content, receiver));
SendInit(sender, content, receiver) =
          ifa ((sender == receiver)||
          (idlist[sender]>idlist[receiver])||
          distancelist[sender][receiver]!=1){ Skip }
          else
          { time!true -> time?x -> {clock = x} ->
            CheckInitTime(clock);
            SendInit2(sender, content, receiver)
          };
SendInit2(sender, content, receiver) =
          {MACInit = call(MACValue,KeySD,msgreq)}->
           broadcast!sender.receiver.content.MACInit->
           {msglist[sender][0] = sender;
            msglist[sender][1] = content;
            msglist[sender][2]=MACInit;
            msglist[sender][3]=0
           } ->Skip;
```

In the above codes, some parameters from the model are defined as the global variables which we have already mentioned before. Here, the process of the whole system is represented:

```
System() = Clock(0) |||( Initiator(1)|||Nodes() ||| Target(3));
```

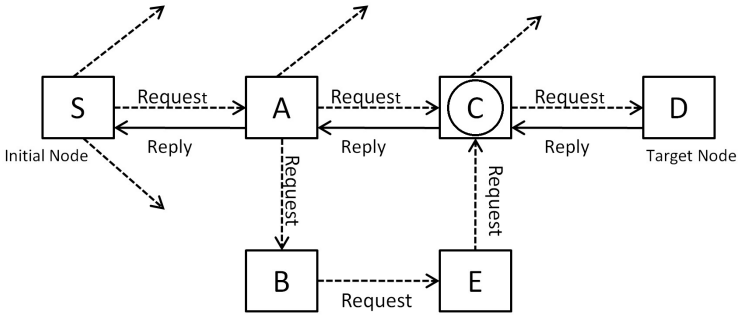

Fig. 4. Case II with Active-1-1 Intruder

Case II: Considering the Active-1-1 intruder, the intruder owns only one compromised node. We add the process of the intruder into the system and the topology is shown in Figure 4, in which we use a circle to identify the intruder node. In the code, the id corresponding to the node (S, A, B, E, C, D) is $(0, 1, 2, 3, 4, 5)$.

When the intruder C receives the message from A, it will wait until the message from E arrives. Then C will change the message especially the node list. After C gets the reply from D, it will cache the reply message until its own TESLA key and the TESLA key of node B have been released, then it will send the reply to A directly. In fact, there is a fake path that S does not recognize. This attack has been mentioned in [5], here we implement it using PAT and we will discuss the details in the subsection Result Analysis. We give the relevant codes about the actions of the intruder in this case, shown below:

```
Intruder(i) = broadcast?sender.i.msg.mac ->
              save.i{intruderlist[j][0]=sender;
              intruderlist[j][1]=i;intruderlist[j][2]=msg;
              intruderlist[j][3]=mac;j=j+1;
              cnt_intruder = cnt_intruder + 1}-> Change(i,j);
              ifa (cnt_intruder >= 2) {Skip}
              else {Intruder(i)};
Change(i,k) = ifa(k==2)
              {change.i{msglist[i-1][0]=i;
               msglist[i-1][2]=
                      call(HashValue,i,msglist[i-2][2]);
               msglist[i-1][3]=
                      call(mediaMACValue,KeyTi[i-1],
                      msglist[i-1][1],msglist[i-1][2],i,
                      msglist[i-2][3])}->
               Sendmedias(i,msglist[i-1][1],msglist[i-1][2]);
               ReceiveReply(i)}
              else {Skip};
```

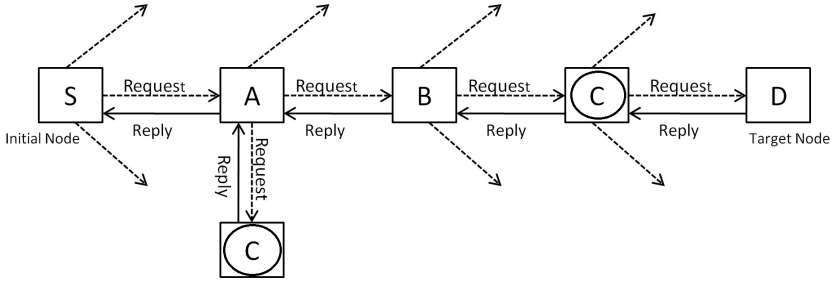

Fig. 5. Case III with Active-1-2 Intruder

Case III: Here, we consider the Active-1-2 intruder, which means that the intruder controls two corrupted nodes but they use the same compromised identifier. The topology is shown in Figure 5 and we use a circle to identify the intruder nodes. The id corresponding to the node (S, A, B, C, D) is $(0, 1, 2, 3, 4)$. When the intruder C receives the request from A, it saves the message into his private list, which the other corrupted node can also access. Then, the intruder whose identifier is also C will receive the message from B, it will change the node list, sending it to D. After C receives the reply message from D, it will also put it into its own list, then the other node whose identity is also named C will send the message back to A ignoring the intermediate node B. Relevant codes are represented below:

```
Intruder(i) = broadcast?sender.i.msg.mac ->
              save.i{intruderlist[j+1][0]=sender;
              intruderlist[j+1][1]=i;intruderlist[j+1][2]=msg;
              intruderlist[j+1][3]=mac} ->Change(i);
Change(i) = change.i{msglist[i-1][0]=i;
            msglist[i-1][2]=
                    call(HashValue,i,msglist[i-1][1]);
            msglist[i-1][3]=
                    call(mediaMACValue,KeyTi[i-1],
                    msglist[i-1][1],msglist[i-1][2],
                    i,intruderlist[0][3])}->
            Sendmedias(i,msglist[i-1][1],msglist[i-1][2]);
            ReceiveReply(i);
```

5.2. Verification

Based on the trace analysis, in our paper, we mainly list five properties as follows:

Property 1: Deadlock Freedom

Deadlock Freedom means that there is no state with no further move except

waiting for some sources occupied by other states. The property of Deadlock freedom can be formalized as follows:

$$\forall i \in (Initiator \cup Node \cup Target) \bullet$$
$$(clock <= TimeMax \wedge i.send == true) \Longrightarrow$$
$$(i.receive == true \wedge num(send) == num(receive))$$

Within the specific time, if there is some node sending message through the channel, there must be some node waiting to receive the message and the number of the sent message is equal to the number of the received message.

Property 2: Message Consistency

Message Consistency stands for no changes happened on the messages in the whole process of the protocol. Here, this property is explained below:

$$\exists m, n \in \mathbb{N} \bullet$$
$$(((request == msglist[m][1]) \Longrightarrow (\forall j\ request == msglist[j][1])) \wedge$$
$$((reply == replylist[n][0]) \Longrightarrow (\forall k\ reply == reply[k][0])))$$

All the request messages, especially the message received by the target node, must be consistent with the request sent by the initiator node. And the reply message received by all the nodes, including the initiator node, is the same with the message sent by the target node.

Property 3: Node List Security

Node List Security represents the security of the node list, that is the node list cannot be changed arbitrarily. We use the first order logic language to describe this property:

$$\forall i, j \in NID \bullet$$
$$(nodelist[i] == requestlist[i][0]) \wedge (replylist[j+1][2] == nodelist[j])$$
where $NID =_{df} 0..N$

The Ariadne protocol aims to prevent a malicious node from compromising the route. It wants to ensure the security of the node list that no intruder can change the order of the nodes, or add, remove any node from the node list. So the initiator receives the node list, which must be the same with that the target node receives.

Property 4: Fake Path Nonexistence

Fake Path Nonexistence is one of the most important properties we discussed in our paper. It means that there exists no fake path in the whole process of the route protocol. This property can be formalized as follows:

$$\forall k \in NID \bullet (nodelist[k] == msglist[k][0]) \wedge$$
$$(distance[msglist[k][0]][msglist[k+1][0] == 1)$$

where $NID =_{df} 0..(N-1)$

If the initiator receives the node list, it will check the distance of each node in the node list. The distance between neighbor nodes being not equal to one means that there may exist a fake path in the node list.

Property 5: End-to-End Nodes Authentication

End-to-End Nodes Authentication means that the end nodes also have some functions to certificate the correctness of the key value and the MAC value. Here, we mainly consider about the authentication of the initiator. This property is explained below:

$$\forall k \in NID \bullet (KeyValue(msgreply[k][1]) == Key_{t_j}[k]) \wedge (MACValue(Key_{SD}, msgreply[k][0]) == MACTar)$$

where $NID =_{df} 0..(N-1)$

In Ariadne protocol, when the initiator receives the reply form the last node, it will check the MAC value of the target and each key in the key list according to the TESLA protocol.

The assertions of these properties can be found in Table 3. According to the order of the properties as mentioned above, the following assertions describe Deadlock Freedom, Message Consistency, Node List Security, Fake Path Nonexistence and End-to-End Nodes Authentication respectively.

Table 3. Assertions of Properties

```
#define goal1 (!((clock ≤ TimeMax)&&(countsend !=0))||
               ((countreceive !=0)&&(countsend==countreceive)));
#assert System() reaches goal1;

#define goal2 ((msgreq==msgReced)&& (msgrep==msgRepReced));
#assert System() reaches goal2;

#define goal3 ((msglist[1][0]==1)&&(msglist[2][0]==2)&& (msglist[3][0]==4));
#assert System() reaches goal3;

#define goal4 (((msglist[1][0]==1)&&(msglist[2][0])==2&&(msglist[3][0]==4))&&
               ((distancelist[msglist[1][0]][0]==1)&&
               (distancelist[msglist[2][0]][msglist[1][0]]==1)&&
               (distancelist[msglist[3][0]][msglist[2][0]]==1)));
#assert System() reaches goal4;

#define goal5 ((call(KeyValue,msgreply[1][1]))==KeyTj[1]&&
               (call(KeyValue,msgreply[0][1]))==KeyTj[0]&&
               (call(MACValue,KeySD,msgreply[1][0]))==MACTar);
#assert System() reaches goal5;
```

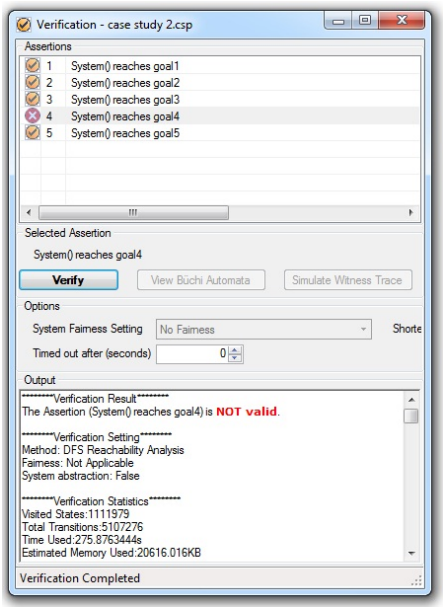

Fig. 6. The Result of Case II

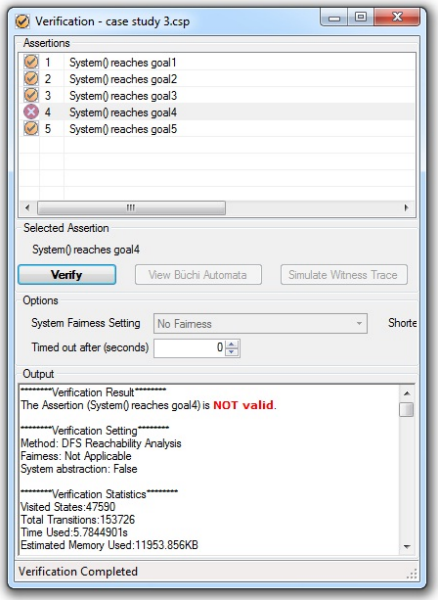

Fig. 7. The Result of Case III

Table 4. The Results of the Verification

	Property 1	Property 2	Property 3	Property 4	Property 5
Case Study 1	P	P	P	P	P
Case Study 2	P	P	P	F	P
Case Study 3	P	P	P	F	P

5.3. Results Analysis

The results of the verification are shown in Table 4. Here, P is the abbreviation of Pass which means the case study reaches the goal and passes the verification of the property. And F, shorted for Fail, means the case does not reach the goal. Through the table, we see that Case I reaches all of the five goals because of no fake path existing in Case I. Conversely, both Case II and Case III fail the verification of property 4 indicating there are fake pathes in these two case studies. Here, we discuss the details about these two results.

In the Case II, when the intruder C receives the message $\langle msg_{req}, h_A, (A), (M_A)\rangle$ from A, it will wait until the message $\langle msg_{req}, h_E, (A, B, E), (M_A, M_B, M_E)\rangle$ from E arrives. Then C will change the message especially the node list as (A, B, C) to D. The modified message is $\langle msg_{req}, h_C, (A, B, C), (M_A, M_B, M_C)\rangle$, where $h_C = H(C, H(B, h_A))$ and M_B can be deduced from the known information. After C gets the reply from D, it will cache the reply message until its own TESLA key and the TESLA key of node B have been released, then it will send the reply to A directly. In fact, (S, A, B, C) is a fake path that S does not

recognize. Meanwhile, we consider the Active-1-2 intruder in Case III. When the intruder C receives the request $\langle msg_{req}, h_A, (A), (M_A)\rangle$ from A, it saves the message into his private list, which the other corrupted node can also access. Then, the intruder whose identifier is also C will receive the message from B, it will change the node list as (A, C) and send it to D. After C receives the reply message from D, it will also put the reply message into its own list, then the other node whose identity is also named C will send the message back to A ignoring the intermediate node B. We also give the User Interface Figure, Figure 6 and Figure 7, which show the result of the verification of Case II and Case III in PAT.

6. Discussion

In Mobile Ad Hoc Networks, typical routing protocols are divided into three types, such as the proactive routing protocols, the on-demand (or reactive) routing protocols and the mixed routing protocols. In our paper, we focus on the on-demand routing ones, i.e. AODV [2,25], DSR [11], Ariadne, etc., in which the initial node will begin the process of route discovery to the destination only when it has a data packet needed to be sent to the destination. Through the comprehensive comparison and analysis, we find that all the on-demand routing protocols have the phase of the route discovery and we think that the modeling approach presented in our paper is also applicable to other protocols, which are based on the on-demand routing protocols and have the route discovery process.

The route discovery includes two processes: broadcasting the request message and unicasting the reply message. We can abstract the process of the route discovery of all the on-demand routing protocols as shown in Figure 8 below:

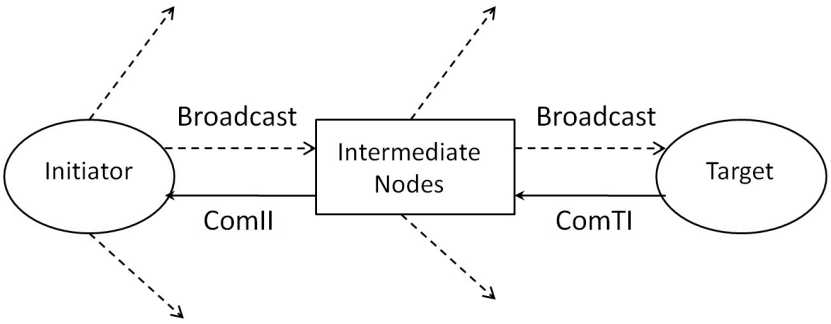

Fig. 8. The Abstract Model Diagram of the Route Discovery Process

We can define sets, i.e., **Initiator**, **Node**, **Target**, **MSG**, and channels such as **Broadcast**, **ComII** and **ComTI**, as mentioned in the previous sections. Some

more sets and channels can be added as needed. We give a general formalization here:

$$
\begin{aligned}
&Initiator(parameter) =_{df} |||_{X_i \in Node \wedge i \in [1,n] \wedge n \in N}(\\
&\quad Broadcast!msg_{req}....Initiator.X_i \rightarrow \\
&\qquad \text{/*More processes can be added here*/} \\
&\quad ComII?msg_{rep}....X_i.Initiator) \rightarrow Initiator(parameter)
\end{aligned}
$$

In the whole discovery process, the initiator will broadcast the request message to all of the nodes only one hop away from it, and receive the reply message from some intermediate node or the target. The actions of the intermediate nodes are more complex, that are receiving the request and the reply messages, broadcasting the request message and unicasting the reply message. Here, we do not discuss the details between different intermediate nodes, considering all the intermediate nodes as a whole part, shown below:

$$
\begin{aligned}
&Node(parameter) =_{df} (|||_{X_n \in Node \wedge n \in N}(\\
&\quad Broadcast?msg_{req}....Initiator.X_n \rightarrow \\
&\qquad \text{/*More processes can be added here*/} \\
&\quad Broadcast!msg_{req}....X_n.Target \rightarrow \\
&\qquad \text{/*More processes can be added here*/} \\
&\quad ComTI?msg_{rep}....Target.X_n \rightarrow \\
&\qquad \text{/*More processes can be added here*/} \\
&\quad ComII!msg_{rep}....X_n.Initiator) \rightarrow Node(parameter)
\end{aligned}
$$

The target node will receive the request message and unicast the reply message to the intermediate nodes. It can be formalized as follows:

$$
\begin{aligned}
&Target(parameter) =_{df} \\
&\quad Broadcast?msg_{req}....X_n.Target \rightarrow \\
&\qquad \text{/*More processes can be added here*/} \\
&\quad ComTI!msg_{rep}....Target.X_n \rightarrow Target(parameter)
\end{aligned}
$$

As mentioned above, we give a general framework to formalize the communication entities as CSP processes of route discovery of the on-demand routing protocols. Some other models, such as intruder model, clock model, etc., can be modeled as needed and the corresponding processes can also be added into our general framework. For instance, in our paper, we focus on the Ariadne protocol, which is an efficient and well-known on-demand secure protocol of MANETs. It mainly concerns about how to prevent a malicious node from compromising the route using three authentication mechanisms. Therefore, we add the security authentication mechanisms into our model and we also define some other processes to describe the security mechanisms.

7. Conclusion and Future Work

In this paper, we proposed a CSP model of the Ariadne secure route protocol. All the communication entities of the protocol, involving the initiator node, the intermediate nodes, and the target, have been abstracted as processes respectively. They share the same clock to realize the effect of asymmetric key encryption through clock synchronization and detention. Besides, we also proposed an intruder model in which the intruder can perform the attacks. We also discuss that our modeling approach is applicable to other on-demand routing protocols, which have the route discovery process. Furthermore, we use PAT, a model checker to implement and validate our formal model automatically. Three case studies are given and we verify five properties: Deadlock Freedom, Message Consistency, Node List Security, Fake Path Nonexistence and End-to-End Nodes Authentication. The verification result demonstrates that some fake paths indeed exist in the Ariadne protocol.

In the future, we would work on the formalizing of the other phase of the Ariadne protocol, including the process of the route maintain. Besides, based on the whole achieved model of the Ariadne protocol, further verification by using PAT is also an interesting topic to be explored. We also want to propose a research methodology to study the discovery process of the on-demand routing protocols for ad hoc networks.

Acknowledgments. The authors gratefully acknowledge support from the Danish National Research Foundation and the National Natural Science Foundation of China (Grant No. 61061130541) for the Danish-Chinese Center for Cyber Physical Systems. This work was also supported by National Basic Research Program of China (No. 2011CB302904), National High Technology Research and Development Program of China (No. 2011AA01 0101 and No. 2012AA011205), National Natural Science Foundation of China (No. 61021 004), and Shanghai Leading Academic Discipline Project (No. B412).

References

1. Ács, G., Buttyán, L., Vajda, I.: Provably Secure On-Demand Source Routing in Mobile Ad Hoc Networks. IEEE Trans. Mob. Comput. 5(11), 1533–1546 (2006)
2. Belding-Royer, E.M., Perkins, C.E.: Multicast Operation of the Ad-Hoc On-Demand Distance Vector Routing Protocol. In: MOBICOM. pp. 207–218 (1999)
3. Bergstra, J.A., Klop, J.W.: Algebra of Communicating Processes with Abstraction. Theor. Comput. Sci. 37, 77–121 (1985)
4. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A Theory of Communicating Sequential Processes. J. ACM 31(3), 560–599 (1984)
5. Buttyán, L., Vajda, I.: Towards provable security for ad hoc routing protocols. In: Proc. 2nd ACM workshop on Security of ad hoc and sensor networks. pp. 94–105. SASN '04, ACM, New York, NY, USA (2004)
6. Chen, C., Dong, J.S., Sun, J., Martin, A.: A verification system for interval-based specification languages. ACM Trans. Softw. Eng. Methodol. 19(4) (2010)

7. Ding, J., Zhu, H., Li, Q.: Formal Specification of Automatic DMARF Based on CSP. In: Engineering of Autonomic and Autonomous Systems (EASe), 2011 8th IEEE International Conference and Workshops on. pp. 32 –39 (4 2011)
8. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
9. Hu, Y.C., Johnson, D.B., Perrig, A.: SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks. Ad Hoc Networks 1(1), 175–192 (2003)
10. Hu, Y.C., Perrig, A., Johnson, D.B.: Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks. Wireless Networks 11(1-2), 21–38 (2005)
11. Johnson, D.B., Maltz, D.A.: Dynamic Source Routing in Ad Hoc Wireless Networks. Mobile Computing pp. 153–181 (1996)
12. Jordan, R., Abdallah, C.: Wireless communications and networking: an overview. Antennas and Propagation Magazine, IEEE 44(1), 185 –193 (2 2002)
13. Lin, C.H., Lai, W.S., Huang, Y.L., Chou, M.C.: Secure Routing Protocol with Malicious Nodes Detection for Ad Hoc Networks. Advanced Information Networking and Applications Workshops, International Conference on 0, 1272–1277 (2008)
14. Liu, Y., Sun, J., Dong, J.S.: An Analyzer for Extended Compositional Process Algebras. In: ICSE Companion. pp. 919–920. ACM (2008)
15. Liu, Y., Sun, J., Dong, J.S.: Analyzing Hierarchical Complex Real-time Systems. In: FSE 2010. pp. 365–366 (2010)
16. Liu, Y., Sun, J., Dong, J.S.: PAT 3: An Extensible Architecture for Building Multi-domain Model Checkers. In: ISSRE. pp. 190–199 (2011)
17. Liu, Z.: The Security Analysis of Routing Protocol for Ad Hoc Networks. Journal of Huangshi Institute of Technology 23(4), 29–33 (8 2007)
18. Lowe, G., Davies, J.: Using CSP to Verify Sequential Consistency. Distributed Computing 12(2-3), 91–103 (1999)
19. Lowe, G., Roscoe, A.W.: Using CSP to Detect Errors in the TMN Protocol. IEEE Trans. Software Eng. 23(10), 659–669 (1997)
20. Luu, A.T., Sun, J., Liu, Y., Dong, J.S., Li, X., Tho, Q.T.: SeVe: automatic tool for verification of security protocols. Frontiers of Computer Science in China 6(1), 57–75 (2012)
21. Luu, A.T., Sun, J., Liu, Y., Dong, J.S., Li, X., Tho, Q.T.: SeVe: automatic tool for verification of security protocols. Frontiers of Computer Science in China 6(1), 57–75 (2012)
22. Mauve, M., Widmer, A., Hartenstein, H.: A survey on position-based routing in mobile ad hoc networks. Network, IEEE 15(6), 30 –39 (Nov/Dec 2001)
23. Mazur, T., Lowe, G.: Counter Abstraction in the CSP/FDR setting. Electr. Notes Theor. Comput. Sci. 250(1), 171–186 (2009)
24. Milner, R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92. Springer (1980)
25. Perkins, C.E., Belding-Royer, E.M.: Ad-hoc On-Demand Distance Vector Routing. In: WMCSA. pp. 90–100 (1999)
26. Perrig, A., Canetti, R., Song, D.X., Tygar, J.D.: Efficient and Secure Source Authentication for Multicast. In: Proc. the Network and Distributed System Security Symposium. The Internet Society (2001)
27. Perrig, A., Canetti, R., Tygar, J.D., Song, D.X.: Efficient Authentication and Signing of Multicast Streams over Lossy Channels. In: IEEE Symposium on Security and Privacy. pp. 56–73 (2000)
28. Pura, M.L., Bica, I., Patriciu, V.V.: On modeling and formally verifying secure explicit on-demand ad hoc routing protocols. In: Proc. 2nd International Conference on Software Technology and Engineering. vol. 2, pp. 215–220 (10 2010)

29. Rohrmair, G.T., Lowe, G.: Using CSP to Detect Insertion and Evasion Possibilities within the Intrusion Detection Area. In: Proc. 1st International Conference on Formal Aspects of Security. pp. 205–220. Springer (2002)
30. Roscoe, A.W.: The theory and practice of concurrency. Prentice Hall (1998), http://www.cs.ox.ac.uk/people/bill.roscoe/publications/68b.pdf
31. Roscoe, A.: Understanding Concurrent Systems. Springer (2010), http://www.comlab.ox.ac.uk/ucs
32. Sanzgiri, K., Dahill, B., Levine, B.N., Shields, C., Belding-Royer, E.M.: A Secure Routing Protocol for Ad Hoc Networks. In: Proc. 10th IEEE International Conference on Network Protocols. pp. 78–89. IEEE Computer Society (2002)
33. Sivakumar, K.A., Ramkumar, M.: Improving the resiliency of Ariadne. In: Proc. 9th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks. pp. 1–6. IEEE (June 2008)
34. Sun, J., Liu, Y., Dong, J.S.: Model Checking CSP Revisited: Introducing a Process Analysis Toolkit. In: Proc. 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. pp. 307–322. Springer (2008)
35. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: Towards Flexible Verification under Fairness. Lecture Notes in Computer Science, vol. 5643, pp. 709–714. Springer (2009)
36. Sun, J., Liu, Y., Dong, J.S., Pu, G., Tan, T.H.: Model-based Methods for Linking Web Service Choreography and Orchestration. In: APSEC 2010. pp. 166 – 175 (2010)
37. Sun, J., Liu, Y., Song, S., Dong, J.S., Li, X.: PRTS: An Approach for Model Checking Probabilistic Real-Time Hierarchical Systems. In: Qin, S., Qiu, Z. (eds.) Formal Methods and Software Engineering. Lecture Notes in Computer Science, vol. 6991, pp. 147–162. Springer Berlin / Heidelberg (2011)
38. Sun, J., Song, S., Liu, Y.: Model Checking Hierarchical Probabilistic Systems. In: Dong, J.S., Zhu, H. (eds.) Formal Methods and Software Engineering - 12th International Conference on Formal Engineering Methods, ICFEM 2010, Shanghai, China, November 17-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6447, pp. 388–403. Springer (2010)
39. Suresh, S., Mike, W., S., R.C.: Power-aware routing in mobile ad hoc networks. In: Proc. 4th annual ACM/IEEE international conference on Mobile computing and networking. pp. 181–190. MobiCom '98, ACM, New York, NY, USA (1998)
40. Tuan, L.A.: Modeling and Verifying Security Protocols Using PAT Approach. Secure Software Integration and Reliability Improvement Companion, IEEE International Conference on 0, 157–164 (2010)
41. Wang, M., Zhu, H., Zhao, Y., Liu, S.: Modeling and Analyzing the (mu)TESLA Protocol Using CSP. In: TASE. pp. 247–250 (2011)
42. Wu, X., Liu, S., Zhu, H., Zhao, Y., Chen, L.: Modeling and Verifying the Ariadne Protocol Using CSP. In: ECBS. pp. 24–32 (2012)
43. Zapata, M.G., Asokan, N.: Securing ad hoc routing protocols. In: Proc. 2002 ACM Workshop on Wireless Security. pp. 1–10. ACM (2002)
44. Zheng, M., Sun, J., Liu, Y., Dong, J.S., Gu, Y.: Towards a Model Checker for NesC and Wireless Sensor Networks. In: Qin, S., Qiu, Z. (eds.) Formal Methods and Software Engineering. Lecture Notes in Computer Science, vol. 6991, pp. 372–387. Springer Berlin / Heidelberg (2011)
45. Zheng, M., Sun, J., Sanán, D., Liu, Y., Dong, J.S., Gu, Y.: Towards bug-free implementation for wireless sensor networks. In: SenSys. pp. 407–408 (2011)

Xi Wu received her BSc in Software Engineering from Software Engineering Institute, East China Normal University in 2011. She is currently a master student in Formal Methods with the same institute. Her research interests include process algebra and its applications, program analysis and verification, and web services.

Huibiao Zhu is Professor of Computer Science at Software Engineering Institute, East China Normal University. He received his BSc in Mathematics and MSc in Computer Science in 1989 and 1992 respectively, all from East China Normal University. He earned his PhD in Formal Methods from London South Bank University in 2005. His research interests include the following areas: (1) semantics theory, including process algebra and its applications; (2) unifying theories of programming; (3) formal design, specification and verification in hybrid systems.

Yongxin Zhao held a PhD degree in Formal Methods from Software Engineering Institute, East China Normal University in 2012. He is currently a research fellow at School of Computing of National University of Singapore. His research interests include program analysis and verification, semantics theory, web services. He owns more than 15 referred publications.

Zheng Wang received his BSc and PhD from Software Engineering Institute, East China Normal University in 2007 and 2012 respectively. Now he is a software requirement engineer in the Software Development Department at Beijing Institute of Control Engineering, China Academy of Space Technology. His main research topic focuses on the automatization and formalization of requirement analysis for embedded control software.

Si Liu received his BSc and MSc from Software Engineering Institute, East China Normal University in 2009 and 2012 respectively. He is currently a PhD student with the Formal Methods and Declarative Languages Laboratory, the Department of Computer Science, University of Illinois at Urbana-Champaign. His research interests are in the areas of formal methods and programming languages.

Received: June 1, 2012; Accepted: August 30, 2012.