Formal verification of secure ad-hoc network routing protocols using deductive model-checking

Levente Buttyán Ta Vinh Thong
Budapest University of Technology and Economics
Laboratory of Cryptography and Systems Security (CrySyS)
Email: {buttyan,thong}@crysys.hu

***Abstract*—Ad-hoc networks do not rely on a pre-installed infrastructure, but they are formed by end-user devices in a self-organized manner. A consequence of this principle is that end-user devices must also perform routing functions. However, end-user devices can easily be compromised, and they may not follow the routing protocol faithfully. Such compromised and misbehaving nodes can disrupt routing, and hence, disable the operation of the network. In order to cope with this problem, several secured routing protocols have been proposed for ad-hoc networks. However, many of them have design flaws that still make them vulnerable to attacks mounted by compromised nodes. In this paper, we propose a formal verification method for secure ad-hoc network routing protocols that helps increasing the confidence in a protocol by providing an analysis framework that is more systematic, and hence, less error-prone than the informal analysis. Our approach is based on a new process calculus that we specifically developed for secure ad-hoc network routing protocols and a deductive proof technique. The novelty of this approach is that contrary to prior attempts to formal verification of secure ad-hoc network routing protocols, our verification method can be made fully automated.**

I. Introduction

In the recent past, the idea of ad-hoc networks have created a lot of interest in the research community, and it is now starting to materialize in practice in various forms, ranging from static sensor networks through opportunistic interactions between personal communication devices to vehicular networks with increased mobility. A common property of these systems is that they have sporadic access, if at all, to fixed, pre-installed communication infrastructures. Hence, it is usually assumed that the devices in ad-hoc networks play multiple roles: they are terminals and network nodes at the same time.

In their role as network nodes, the devices in ad-hoc networks perform basic networking functions, most notably routing. At the same time, in their role as terminals, they are in the hand of end-users, or they are installed in physically easily accessible places. In any case, they can be easily compromised and re-programmed such that they do not follow the routing protocol faithfully. The motivations for such re-programming could range from malicious objectives (e.g., to disrupt the operation of the network) to selfishness (e.g., to save precious resources such as battery power). The problem is that such compromised and misbehaving routers may have a profound negative effect on the performance of the network.

In order to mitigate the effect of misbehaving routers on network performance, a number of secured routing protocols have been proposed for ad-hoc networks (see e.g., [11] for a survey). These protocols use various mechanisms, such as cryptographic coding, multi-path routing, and anomaly detection techniques, to increase the resistance of the protocol against attacks. Unfortunately, the design of secure routing protocols is an error-prone activity, and indeed, most of the proposed secure ad-hoc network routing protocols turned out to be still vulnerable to attacks. This fact implies that the design of secure ad-hoc network routing protocols should be based on a systematic approach that minimizes the number of mistakes made in the design.

As an important step towards this goal, in this paper, we propose a formal method to verify the correctness of secure ad-hoc network routing protocols. Our approach is based on a new process calculus that we specifically developed for modeling the operation of secure ad-hoc network routing protocols, and a proof technique based on deductive model checking. The systematic nature of our method and its well-founded semantics ensure that one can have much more confidence in a security proof obtained with our method than in a ”proof” based on informal arguments. In addition, compared to previous approaches that attempted to formalize the verification process of secure ad-hoc network routing protocols [8], [2], [3], [4], the novelty of our approach is that it can be fully automated.

The organization of the paper is the following: In Section II, we describe the SRP protocol [13] as an example for a secure ad-hoc network routing protocol that we will use for illustration purposes throughout the paper. In Section III, we introduce the syntax and the semantics of our process calculus. In Section IV, we demonstrate the expressive power of our calculus by modeling the operation of SRP and by formally representing a known security flaw in SRP. In Section V, we discuss how the verification process can be automated and describe our deductive proof technique. Finally, in Section VI, we conclude the paper and discuss our planned future work on this topic.

II. The SRP protocol and an attack on it

SRP is a secure on-demand source routing protocol for ad-hoc networks proposed in [13]. The design of the protocol is inspired by the DSR protocol [12], however, DSR has no security mechanisms at all. Thus, SRP can be viewed as a secure variant of DSR. SRP tries to cope with attacks by using a cryptographic checksum in the routing control messages

(route requests and route replies). This checksum is computed with the help of a key shared by the initiator and the target of the route discovery process; hence, SRP assumes only shared keys between communicating pairs.

In SRP, the initiator of the route discovery generates a route request message and broadcasts it to its neighbors. The integrity of this route request is protected by a Message Authentication Code (MAC) that is computed with a key shared by the initiator and the target of the discovery. Each intermediate node that receives the route request for the first time appends its identifier to the request and re-broadcasts it. The MAC in the request is not updated by the intermediate nodes, as by assumption, they do not necessarily share a key with the target. When the route request reaches the target of the route discovery, it contains the list of identifiers of the intermediate nodes that passed the request on. This list is considered as a route found between the initiator and the target.

The target verifies the MAC of the initiator in the request. If the verification is successful, then it generates a route reply and sends it back to the initiator via the reverse of the route obtained from the route request. The route reply contains the route obtained from the route request, and its integrity is protected by another MAC generated by the target with a key shared by the target and the initiator. Each intermediate node passes the route reply to the next node on the route (towards the initiator) without modifying it. When the initiator receives the reply it verifies the MAC of the target, and if this verification is successful, then it accepts the route returned in the reply.

The basic problem in SRP is that the intermediate nodes cannot check the MAC in the routing control messages. Hence, compromised intermediate nodes can manipulate control messages, such that the other intermediate nodes do not detect such manipulations. Furthermore, the accumulated node list in the route request is not protected by the MAC in the request, hence it can be manipulated without the target detecting such manipulations.

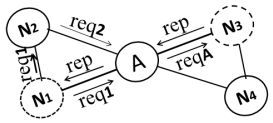

Fig. 1. An attack scenario against the SRP protocol.

In order to illustrate a known attack on SRP, let us consider the network topology shown in Figure 1. Let us further assume that node N_1 initiates a route discovery to node N_3. The attacker node A can manipulate the accumulated list of node identifiers in the route request such that N_3 receives the request with the list (N_2, n, N_4), where n is an arbitrary fake identifier. This manipulation remains undetected, because the MAC computed by N_1 does not protect the accumulated node list in the route request, and intermediate nodes do not authenticate the request. When the target N_3 sends the route reply, A forwards it without modification to N_1 in the name of N_2. As the route reply is not modified, the MAC of the target N_3 verifies correctly at N_1, and thus, N_1 accepts the route (N_1, N_2, n, N_4, N_3). However, this is a mistake, because there is no link between N_2 and n, and between n and N_4.

Note that the above attack has been found by manual analysis of the protocol. However, there may be many similar attack scenarios (and indeed, there are), and manual analysis would be inefficient to find all of them. The very purpose of our formal verification method to be introduced in the upcoming sections of the paper is to make the analysis systematic and amenable for automation such that it can efficiently find all possible attacks against a protocol (within some limits of the underlying model).

III. Our Calculus

In this section we define our calculus: its syntax and informal semantics, as well as its operational semantics.

The advantage of our calculus is that its expressiveness allows for modelling broadcast communication, neighborhood, and transmission range like CMAN [10] and the ω-calculus [14], and cryptographic primitives like the applied π-calculus and the spi-calculus [9], [1], however, compared to them it includes the definition of *active substitution with range* that is novel and enables us to model attacker knowledge and attacks in the context of wireless ad-hoc networks. More details about the novelties can be found in the technical report [7].

A. Syntax

We assume an infinite set of *names* $\mathcal{N}$ and *variables* $\mathcal{V}$, where $\mathcal{N} \cap \mathcal{V} = \emptyset$. Further, we define the set of *node identifiers* (node ID) $\mathcal{L}$, where $\mathcal{N} \cap \mathcal{L} = \emptyset$. Each node identifier uniquely identifies a node.

We define *term* t as an element of the set $\{req,\ rep,\ c,\ n,\ l,\ x,\ f(t_1, \ldots, t_k)\}$: *req* and *rep* are unique constants that represent the *req* and *rep* tags in route request and reply messages; c models communication channel; n models some data; l models a node ID, and x is a variable that models any term. Finally, f is a function with arity k and is used to model cryptographic primitives, and route request and reply messages. For instance, digital signature is modelled by the function *sign*(t_1, t_2), where t_1 models the message to be signed and t_2 models the secret key. Route request and reply messages are modelled by the function *tuple* of k terms *tuple*$(t_1,\ldots,t_k)$, which we abbreviate as $(t_1,\ldots,t_k)$.

The internal operation of nodes is modelled by *processes*. Processes can be inductively specified with the following syntax:

- The processes $\overline{c}\langle t\rangle.P$, $c(x).P$ represent unicast of message t on communication channel c, and followed by the process P.
- The two processes $\langle t\rangle.P$, and $(x).P$ represent the broadcast send and receive of t continued with process P. Compared to the unicast case it does not contain the channel, which intends to model that there is not a specified target.
- Processes $[t_1 = t_2]P$ and $[l \in \sigma]P$ mean that if $t_1 = t_2$ and $l \in \sigma$, respectively, then process P is "activated", else it stucks and idle.

- Process *let* $t = x$ *in* P represents the binding of term t to variable x in process P.
- Process $P|Q$ is the *parallel composition* of processes P and Q and behaves as processes P and Q running in parallel: each may interact with the other, or with the outside world, independently of the other.
- Finally, process $!P$ represents the *infinite replication* of process P.

Nodes are defined as $\lfloor P \rfloor_l^\sigma$, which represents a node that has the identifier l and behaves as P, and its transmission range covers the nodes whose identifiers are in the set σ. Two nodes are neighbors if they are in each other's range. We note that σ can be empty, denoted as $\lfloor P \rfloor_l$, which means that the node has no connections.

A network, denoted as N, can be an empty network with no node: 0_N; a singleton network with one node: $\lfloor P \rfloor_l$; and the parallel composition of networks: $N_1|N_2$.

Finally, in order to model attackers that improves their knowledge by accumulating information it hears from their neighbors, we extend the syntax of networks with the *active substitution with range*. An *extended network* E is equipped with the active substitution $\{t/x\}^\sigma$, which models that the substitution $\{t/x\}$ binds t to variable x that occurs in any node $\lfloor P \rfloor_{l_i}^{\sigma_i}$ that is in parallel composition with $\{t/x\}^\sigma$ and $l_i \in \sigma$. Intuitively, σ is the range of the substitution $\{t/x\}$.

B. Semantics

The operational semantics of our calculus is defined by the labelled transition system (LTS) detailed in [7]. Due to page limitation only five rules are discussed here:

(Br-snd) $\lfloor \langle t \rangle.P \rfloor_l^\sigma \xrightarrow{\nu x.\langle x \rangle:\bar{l}\sigma} \left(\{t/x\}^\sigma | \lfloor P \rfloor_l^\sigma\right)$

(Br-rcv) $\left(\{t/x\}^\sigma | \lfloor (x).Q \rfloor_l^{\sigma_2}\right) \xrightarrow{(t):\{l \in \sigma\}} \left(\{t/x\}^\sigma | \lfloor Q\{t/x\} \rfloor_l^{\sigma_2}\right)$

(Equal-Red) $\lfloor [t_1 = t_2].P \rfloor_l^\sigma \longrightarrow \lfloor P \rfloor_l^\sigma$, if $t_1 = t_2$.

(In-Red) $\lfloor [l \in \sigma].P \rfloor_l^\sigma \longrightarrow \lfloor P \rfloor_l^\sigma$, if $l \in \sigma$.

(Let-Red) $\lfloor let\ t = x\ in\ P \rfloor_l^\sigma \longrightarrow \lfloor P\{t/x\} \rfloor_l^\sigma$

The first two rules are *labelled transitions*. The first rule means that node l broadcasts t, so that t is now available for the nodes in its range σ. This is modelled by $\{t/x\}^\sigma$ and νx, which restricts the substitution to the nodes that are within the range σ. The second rule means that if the listening node l is within the range σ then it receives the broadcasted t. The third, fourth, and fifth rules are *internal reduction* rules that model the internal computation of nodes: the third rule models the equality check of two terms; the fourth rule checks if the node ID l is in the set σ; and the fifth rule models the binding of t to x in P.

C. Labelled bisimilarity in context of wireless ad-hoc networks

In this subsection we give the definition of labelled bisimilarity that tells if two wireless ad-hoc networks are equivalent, meaning that they cannot be distinguished by an observer which can eavesdrop on communications.

Let the extended network E be $(\{^{t_1}/_{x_1}\}^{\sigma_1} \mid \ldots \mid \{^{t_n}/_{x_n}\}^{\sigma_n} \mid N_1 \mid \ldots \mid N_n)$. The frame φ of E is the parallel composition $\{^{t_1}/_{x_1}\} \mid \ldots \mid \{^{t_n}/_{x_n}\}$ that models all of the information that is output so far by the network E, which is $t_1, \ldots, t_n$ in this case.

We say that two networks E_1 and E_2 are *statically equivalent*, denoted as $E_1 \approx_s E_2$, if their frames are statically equivalent. Two frames φ_1, φ_2 are statically equivalent if they includes the same number of active substitutions and same domain; and any two terms that are equal in φ_1 are equal in φ_2 as well. Intuitively, this means that the outputs of the two networks cannot be distinguished.

Definition 1. *Labelled bisimilarity ($\approx_l^N$) is the largest symmetric relation $\mathcal{R}$ on closed extended networks such that $E_1 \mathcal{R} E_2$ implies: $\mathcal{L}(E_1) = \mathcal{L}(E_2)$, $\mathcal{C}(E_1) = \mathcal{C}(E_2)$, if*

- $E_1 \approx_s E_2$*;*
- *if $E_1 \rightarrow E_1'$, then $E_2 \rightarrow^* E_2'$ and $E_1' \mathcal{R} E_2'$ for some E_2';*
- *if $E_1 \xrightarrow{\nu x.\langle x \rangle:\bar{l}\sigma} E_1'$, then $\exists E_2'$ such that $E_2 \rightarrow^* \xrightarrow{\nu x.\langle x \rangle:\bar{l}\sigma} \rightarrow^* E_2'$ and $E_1' \mathcal{R} E_2'$ for some E_2'.*

Intuitively, this means that the outputs of the two networks of same topology cannot be distinguished during their operation. $\mathcal{L}(E)$ and $\mathcal{C}(E)$ are the set of node IDs, and the neighborhood in E, respectively. In particular, the first point means that at first E_1 and E_2 are statically equivalent; the second point says that E_1 and E_2 remains statically equivalent after internal reduction steps. Finally, the third point says that if the node l in E_1 outputs something then the node l in E_2 outputs the same thing, and the "states" E_1' and E_2' they reach after that remain statically equivalent. Here, $\rightarrow^*$ models the sequential execution of some internal reductions.

IV. Application of our calculus

Next we demonstrate the usabiliy of our calculus by modelling the SRP protocol and the attack showed in Section II. The scenario in Section II is modelled by the extended network defined as:

$$netw \stackrel{def}{=} (\lfloor P_1 \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor P_2 \rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}}).$$

where the description of the nodes in the parallel compositions above corresponds to N_1, N_2, A, N_3, and N_4, respectively.

We use the following functions: The $mac(t_1, t_2)$ function computes the message authentication code of the message t_1 using the secret key t_2. The shared key between the nodes l_i and l_j is modelled by the function $k(l_i, l_j)$. The function $[l_1, \ldots, l_n]$ models the list of node IDs. Functions *prev*$(List, l_i)$ and *next*$(List, l_i)$ return the element right before and after l_i in the list *List*, respectively; they return *undef* if there is no any element before or after l_i. Function *toend*$(List, l_i)$ that we abbreviate with $[List, l_i]$ appends l_i to the end of *List*. Functions *fst*(*List*), *lst*(*List*) represent the first, and last element of *List*, respectively. Function $i((t_1, \ldots, t_n))$ returns the i-th ($i \in \{1, \ldots n\}$) element t_i of the tuple $(t_1, \ldots, t_n)$. Further, we extend terms with the constant *RouteOK*. The source node outputs *RouteOK* when it receives the reply message and all the verifications it makes on it are successful.

The operation of the source node is modelled as follows[1]:

$$
\begin{aligned}
P_1 &\stackrel{def}{=} \textit{let } MAC_{13} = mac\left((l_1, l_3), k(l_1, l_3)\right) \textit{ in ReqInit.}\\
\textit{ReqInit} &\stackrel{def}{=} \langle(req, l_1, l_3, MAC_{13}, [\,])\rangle.!\textit{WaitRep}_1.\\
\textit{WaitRep}_1 &\stackrel{def}{=} (x_{rep}).[1(x_{rep}) = l_1]\ [2(x_{rep}) = rep]\ [3(x_{rep}) = l_1]\\
&\quad [4(x_{rep}) = l_3][fst(5(x_{rep})) \in \{l_2 l_a\}]\\
&\quad [mac\left(\left(l_1, l_3, 5(x_{rep})\right), k(l_1, l_3)\right) = 6(x_{rep})]\\
&\quad \langle \textit{RouteOK}\rangle.
\end{aligned}
$$

Intuitively, the first row models that the node l_1 computes the MAC using the key it shares with the detination node l_3. The second row means that node l_1 generates the route request message that includes the ID of the source and the target nodes, and the message authentication code MAC_{13}, then l_1 broadcasts it and waits for the reply. The exclamation mark models the infinite replication of $\textit{WaitRep}_1$. Finally, the third row means that when l_1 receives a message, it checks whether (i) it is the addressee, (ii) the message is a reply, (iii) the ID of the source and the target nodes, and (iv) the message authentication code are correct. If all are correct then it signals the special constant *RouteOK*.

The description of the process P_2 is the following:

$$
\begin{aligned}
P_2 &\stackrel{def}{=} (y_{req}).[1(y_{req}) = req].\\
&\quad \langle(1(y_{req}), 2(y_{req}), 3(y_{req}), 4(y_{req}), [5(y_{req}), l_2])\rangle\\
&\quad !\textit{WaitRep}_2.\\
\textit{WaitRep}_2 &\stackrel{def}{=} (y_{rep}).[1(y_{rep}) = l_2][2(y_{rep}) = rep]\\
&\quad [next\,(5(y_{rep}), l_2) \in \{l_1 l_a\}]\\
&\quad \langle(l_1, 2(y_{rep}), 3(y_{rep}), 4(y_{rep}), 5(y_{rep}), 6(y_{rep}))\rangle.
\end{aligned}
$$

Intuitively, on receiving a message the node l_2 checks if it is a request, if so then node l_2 appends its ID to the end of the list, re-broadcasts it and waits for a reply. When it receives the reply message it checks (i) if the message is intended to it, (ii) it is a reply, (iii) the next ID in the list corresponds to neighbors and forwards the message to the source node l_1.

Finally, the operation of the destination node is modelled as:

$$
\begin{aligned}
P_3 &\stackrel{def}{=} (z_{req}).[1(z_{req}) = req][3(z_{req}) = l_3].\\
&\quad [mac\left(\langle 2(z_{req}), 3(z_{req})\rangle, k(l_1, l_3)\right) = 4(z_{req})]\ \textit{let } MAC_{31} =\\
&\quad mac((1(z_{req}), 2(z_{req}), 3(z_{req}), 5(z_{req})), k(l_1, l_3))\ \textit{in}\\
&\quad \textit{let } l_{prev} = lst\,(5(z_{req}))\ \textit{in}\\
&\quad \langle(l_{prev}, rep, 2(z_{req}), 3(z_{req}), 5(z_{req}), MAC_{31})\rangle.
\end{aligned}
$$

Intuitively, on receiving the a message node l_3 checks (i) if the message is a request, and (ii) it is the destination, and verifies the MAC embedded in the request using its shared key with l_1. If so then node l_3 creates a reply message and sends it back to the last node in the list.

The description of the node l_4 is the same as the node l_2 with the difference that it appends l_4 to the list instead of l_2. We refer the reader to [7] for details.

We give the model ($\mathcal{M}_A$) of the attacker node as follows: We assume that the attacker cannot forge the message authentication codes MAC_{13} and MAC_{31} without possessing correct keys. Initially, the attacker node knows the IDs of its neighbors $\{l_1, l_2, l_3, l_4\}$. The attacker can create new data n, and can append elements of $\{l_1, l_2, l_3, l_4\}$, and n to the end of the ID list it receives. Finally, it can broadcast and unicast its messages to honest nodes.

The attacker overhears only messages sent by its neighbors. Let the frame $\varphi(l_a)$ be $\{t_i\,/_{x_i}\}^{\sigma_i}|\ \{t_j\,/_{x_j}\}^{\sigma_j}|\ldots|\ \{t_k\,/_{x_k}\}^{\sigma_k}$, where $l_a \in \sigma_i$, $l_a \in \sigma_j$,..., $l_a \in \sigma_k$. This represents the attacker's knowledge he accumulates during the route discovery phase by eavesdropping. He combines this accumulated knowledge and its initial knowledge to construct attacks. Let T_{l^p} be a tuple that consists of the elements in $\{l_1, l_2, l_3, l_4\}$.

Formally, the operation of the attacker node is defined as follows: $P_A \stackrel{def}{=} (\tilde{x})\,.\nu n.\langle f\,(\tilde{x}, T_{l^p}, n)\rangle$, where $\tilde{x}$ is a tuple $(x_1,\ldots,x_m)$ of variables, νn means the attacker creates new data n. The function $f\,(\tilde{x}, T_{l^p}, n)$ represents the message the attacker generates from (i) the eavesdropped messages that it receives by binding them to $\tilde{x}$, (ii) its initial knowledge and (iii) the newly generated data n, respectively.

As the next step, we define an ideal model of *netw*, denoted as $netw_{spec}$. The definition of $netw_{spec}$ is the same as *netw* except that the desription of N_1 is $\lfloor P_1^{spec} \rfloor_{l_1}^{\{l_2 l_a\}}$. Process P_1^{spec} models the ideal operation of the source node N_1 in the sense that although the source node does not know the route to the destination it is equipped with the special function $consistent(List)$ that informs it about the correctness of the returned route.

We define this ideal source node as follow:

$$
\begin{aligned}
P_1^{spec} &\stackrel{def}{=} \textit{let } MAC_{13} = mac\left((l_1, l_3), k(l_1, l_3)\right) \textit{ in ReqInit}_{spec}.\\
\textit{ReqInit}_{spec} &\stackrel{def}{=} \langle(req, l_1, l_3, MAC_{13}, [\,])\rangle.!\textit{WaitRep}_{spec}.\\
\textit{WaitRep}_{spec} &\stackrel{def}{=} (x_{rep}).[1(x_{rep}) = l_1]\ [2(x_{rep}) = rep][3(x_{rep}) = l_1]\\
&\quad [4(x_{rep}) = l_3][fst(5(x_{rep})) \in \{l_2 l_a\}]\\
&\quad [mac\left(\left(l_1, l_3, 5(x_{rep})\right), k(l_1, l_3)\right) = 6(x_{rep})]\\
&\quad [consistent\,(5(x_{rep})) = true].\langle \textit{RouteOK}\rangle.
\end{aligned}
$$

Intuitively, in the ideal model, every route reply that contains a non-existent route is caught and filtered out by the initiator of the route discovery. Hence, in the ideal model security is achieved by definition.

Next we give the *definition of secure routing* based on labelled bisimilarity:

Definition 2. *A routing protocol is said to be secure if for all extended networks E and its corresponding ideal network E_{spec} where both include the same but arbitrary attacker node, we have: $E \approx_l^N E_{spec}$.*

Theorem 1. *The SRP protocol is insecure.*

Proof: **(Sketch)** We will show that $netw \approx_l^N netw_{spec}$ does **not** hold in the presence of the attacker $\mathcal{M}_A$ because the third point of the Definition 1 is violated. When the attacker node receives the request message (req, l_1, l_3, MAC_{13},[l_2]) from node l_2, it creates some new fake node indentifier n, then it adds n and the identifier l_4 to the list [l_2], and re-broadcasts (req, l_1,l_3,MAC_{13},[l_2, n, l_4]). When this message reaches the target node l_3 it passes all the verifications made by l_3. Then, node l_3 generates the reply (l_4, rep, l_1, l_3,MAC_{31}) and sends it back to l_4. The attacker node overhears this message and forwards it to the source l_1 in the name of l_2. As the result, in *netw* node l_1 accepts the returned invalid route $[l_2, n, l_4]$ and outputs *RouteOK* by the $\stackrel{\nu x.\langle x\rangle:\overline{l_1}\{l_2 l_a\}}{\longrightarrow}$ transition relation.

[1]Due to page limitation we omit the presence of sequence number and message ID in messages, but we note that our attack works in the same way in the presence of them.

However, in $netw_{spec}$ node l_1 does not accept the returned route, thus, *RouteOK* is not output. Formally, at this point $netw_{spec}$ cannot perform the transition $\xrightarrow{\nu x.\langle x\rangle:\overline{l_1}\{l_2 l_a\}}$, which violates the third point of Definition 1. ■

V. On automating the verification

In this section, we present a novel automated verification technique based on deductive model checking. The novelty of our method compared to other related works is that it supports explicit modelling of cryptographic primitives which is not the case in [5], [14]. Compared to [8], [2] our method can be fully automated. Our technique was inpired by the concept of the Proverif automatic verification tool [6], however, as opposed to [6] it is designed for verifying routing protocols and includes numerous novelties such as broadcast communications, neighborhood, transmission range, and considers an attacker model specific to wireless ad hoc networks.

A. From the process calculus to Horn-clauses

In the ProVerif verification tool [6] the input of the tool is the protocol description in the syntax of the applied π-calculus. The tool then translates this to Prolog rules to make automatic reasoning. Following this concept, in our verifcation method, routing protocols are specified in the syntax of our calculus. This is then translated to Horn-clauses using translation rules. The reader can find details in [7].

Terms t in the calculus are translated to *patterns* p of Horn-clauses that is defined as follows: Variables x and node IDs l in the calculus are translated to x^p and l^p of Horn-clauses, respectively. The attacker node has the unique identifier l^p_{att}. Node IDs l and l^p_{att} are atomic values. Data n and function f are encoded as functions with arity k and t: $n[p_1,\ldots,p_k]$, and $f(p_1,\ldots,p_t)$, respectively.

In addition, our method uses the *facts* $wm(p)$ and $att(p)$ which model that the wireless medium and the attacker knows pattern p, respectively. The fact $route(l^p_i, l^p_j)$ is used to assign the source node l^p_i and the target node l^p_j for a given route discovery. Finally, the fact $nbr(l^p_i, l^p_j)$ models that node l^p_j is in the transmission range of node l^p_i.

B. Generating protocol clauses

We exploit the fact that each node behaves in the same way in a route discovery phase of routing protocols. Hence, the operation of every node can be uniformly defined as follows:

$$P_i \overset{def}{=} !^{i_1} ReqInit^{P_i} \mid !^{i_2} Interm^{P_i} \mid !^{i_3} Dest^{P_i}$$

Every node can start a route discovery towards any other node, in this case the *ReqInit* process is invoked. Every node can be an intermediate node, in which case its $Interm$ process is invoked. Finally, every node can be a target node when the *Dest* process is invoked. We note that the specified structure of each process depends on the specified routing protocol.

However, in general form P_i can be modelled as:

$ReqInit^{P_i} \overset{def}{=} c_{P_i}(x^p_{dest}).\ ProtDep_{reqinit}.\ \langle x^p_{msgreq}\rangle.!^{i_4} WaitRep^{P_i}_{init}.$

$WaitRep^{P_i}_{init} \overset{def}{=} (x^p_{from}, x^p_{rep}).Verif_{init}.\langle x^p_{msgrep}\rangle.$

$Interm^{P_i} \overset{def}{=} (y^p_{from}, y^p_{req}).ProtDep_{inter}.\langle y^p_{msgreq}\rangle.\ !^{i_5} WaitRep^{P_i}_{inter}.$

$WaitRep^{P_i}_{inter} \overset{def}{=} (y^p_{from}, y^p_{rep}).Verif_{inter}.\langle y^p_{msgrep}\rangle.$

$Dest^{P_i} \overset{def}{=} (z^p_{from}, z^p_{req}).ProtDep_{dest}.\langle z^p_{msgrep}\rangle.$

In short, the process $ReqInit^{P_i}$ receives the ID of the destination node to which it starts the route discovery. Then it goes through a protocol dependent processing part and broadcasts some message and waits for the reply at the end. On receiving some message, process $WaitRep^{P_i}_{init}$ does verification steps in a protocol dependent manner and then broadcasts some message. The other three processes are interpreted in the same way. Patterns x^p_{from}, y^p_{from}, and z^p_{from} are identifiers of the nodes from which the route request and reply messages y^p_{req}, y^p_{rep}, z^p_{req}, and x^p_{rep} are received. Finally, i is a session ID to distinguish different process instances in replication, and is used to track attack traces.

P_i is then translated to the following three kinds of rules. We note that the set of all specified translated rules of the description above can be found in [7]. Here we introduce only the sketch of them:

(General rules used for the protocol) ::=

R_1. $route(x^p_{src}, x^p_{dest}) \overset{\tilde{p},i}{\rightarrow} wm(x^p_{src}, p)$

R_2. $wm(y^p_{from}, p) \wedge nbr(y^p_{from}, y^p_{to}) \overset{p,i}{\rightarrow} wm(y^p_{to}, p')$

R_3. $wm(z^p_{from}, p) \wedge nbr(z^p_{from}, l^p_{att}) \overset{p,i}{\rightarrow} att(p)$.

These rules model the communications from the wireless medium point of view. Rule R_1 initiates the route discovery with a source-destination pair, the label $\tilde{p}$ is the tuple *(req,*x^p_{src}*,*x^p_{dest}*,ID)*, and i is a session ID. R_2 represents the scenario in which node y^p_{to} receives the message p from node y^p_{from} and broadcasts some message p' after processing the received p. Rule R_3 concerns the case in which the attacker node l^p_{att} eavesdrops the messages broadcasted by its neighbors.

C. Knowledge and computation ability of the attacker

The ability of an attacker node is represented in the following rules. These rules represent the strongest actions that can be performed by the attacker node l^p_{att}.

Initially, the attacker knows its own ID and the ID of its honest neighbors: $\forall l^p_n$ neighbors of l^p_{att}, $att(l^p_{att})$, $att(l^p_n)$; it knows all the keys it shares with the honest nodes: $\forall i$, $att(k^p_{att,i})$; and it is aware of its neighborhood: $nbr(l^p_{att}, l^p_n)$;

In addition, we define a strong computation ability for the attacker node as follows:

(Computation ability) ::=

C_1. $att(i) \rightarrow att(n[i])$

C_2. *For each public function* f *of n-arity*
$att(x^p_1) \wedge \ldots att(x^p_n) \rightarrow att(f(x^p_1,\ldots,x^p_n))$

C_3. *For each public function* g *that* $g(f(x^p_1,\ldots,x^p_n), y^p) \rightarrow x^p$:
$att(f(x^p_1,\ldots,x^p_n)) \wedge att(y^p) \rightarrow att(x^p)$

C_4. $att(p) \wedge nbr(l^p_{att}, l^p_i) \rightarrow wm(l^p_i, p')$.

Rule C_1 means that the attacker node can create arbitrary number of new data n. Rule C_2 means that the attacker can generate arbitrary messages based on its actual knowledge. Rule C_3 means that the attacker can perform computation on function f. For instance, if f is a digital signature *sign* then its corresponding "inverse" function g, which is *checksign* can be performed to verify the signature. The set of functions depends

on the protocol we are examining. Finally, the rule C_4 means that if the attacker node has created some message p, it can broadcast it to its neighbors who then output some message p' after receiving p.

D. Deductive algorithm

The operation of a routing protocol is modelled by resolution steps, defined as follows:

Definition 3. *Given two rules r_1=$H_1 \rightarrow C_1$, and r_2= $F \wedge H_2 \rightarrow C_2$, where F is **any** hypothesis of R_2, and F is unifiable with C_1 with the most general unifier σ, then the resolution $r_1 \circ_F r_2$ of them yields a new rule $H_1\sigma \wedge H_2\sigma \rightarrow C_2\sigma$.*

Let us recall the example in Figure 1. The resolution *route*$(l_1^p, l_3^p) \circ_F R_1$, where *p*=(*req*, x_{src}, x_{dest}, *ID, MAC*), and F= *route*(x_{src}^p, x_{dest}^p) yields the fact *wm*((*req*, l_1^p, l_3^p, *ID, MAC*)) with the unifier $\{x_{src}^p \leftarrow l_1^p, x_{dest}^p \leftarrow l_3^p\}$. This resolution step models the broadcasting of the route request message generated by the node l_1^p.

In the following we give a sketch of the deductive algorithm of our method.

Algorithm 1. *The input of the algorithm includes the set of protocol rules $\mathcal{T}_0$; a network topology given by the set of facts nbr(l_i^p, l_j^p); the rules of attacker computation ability, the (initial) knowledge of the attacker: $\mathcal{A}$, $\mathcal{K}$, respectively; and the fact route(l_{src}^p, l_{dest}^p) that specifies the source and target nodes.*

The algorithm is based on the breadth-first search to reach the destination node l_{dest}^p from the source l_{src}^p. Each node broadcasts its message to neighbors. This is modelled by resolution steps at each node. The algorithm starts with the resolution of route(l_{src}^p, l_{dest}^p) and R_1, and followed by the resolutions of rules in $\mathcal{T}_0$ and the conclusions resulted in the previous resolutions. These resolution steps model the communication between intermediate nodes.

When the attacker node intercepts a message broadcasted by one if its neighbors (modeled by the rule R_3) it adds the new fact att(p) into $\mathcal{K}$. This models the accumulated knowledge of the attacker. The attacker node uses rules in $\mathcal{A}$ and $\mathcal{K}$ to construct an incorrect message and forwards it to honest nodes.

Theorem 2. *If there is an attack for a given network topology $\mathcal{N}$ and tuple (l_s^p, l_d^p, l_{att}^p) then the algorithm will find some attack.*

Theorem 3. *The worst case complexity of the algorithm is $\mathcal{D}(|N| + |E|) + \mathcal{B}$, where $|N|$, $|E|$ are the number of nodes and edges of the topology, and $\mathcal{D}$, $\mathcal{B}$ are the number of incorrect messages the attacker can create that are accepted by its honest neighbors, and the number of resolution steps (computation steps) made by the attacker, respectively. For practical reasons both $\mathcal{D}$ and $\mathcal{B}$ are assumed to be finite.*

We note that in a general case this can be exponential, however, by taking into account the property of the specified protocols and properly restricting the attacker ability this complexity can be reasonably reduced.

VI. Conclusion

We argued that designing secure ad-hoc network routing protocols requires a systematic approach which minimizes the number of mistakes made during the design. To this end, we proposed a formal verification method for secured ad-hoc network routing protocols, which is based on a novel process calculus and a deductive proof technique. Our method has a clear syntax and semantics, and it can be fully automated; this latter being a distinctive feature among other formal approaches for verifying secure ad-hoc network routing protocols.

The work described in this paper is work in progress, and we are currently extending it in many ways. The two most important future work items are (i) the development of a fully automated protocol verification software tool based on the theoretical foundations described in this paper, and (ii) the extension of the described verification method to handle arbitrary network topologies and arbitrary number of attacker nodes.

Acknowledgment

The work described in this paper has been supported by the High Speed Networks Laboratory (HSN Lab) at the Budapest University of Technology and Economics. The first author has been further supported by the Hungarian Academy of Sciences through the Bolyai János Research Fellowship.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the Spi calculus. Technical Report SRC RR 149, Digital Equipment Corporation, Systems Research Center, January 1998.

[2] G. Acs, L. Buttyan, and I. Vajda. Provable security of on-demand distance vector routing in wireless ad hoc networks. In *In In Proceedings of the Second European Workshop on Security and Privacy in Ad Hoc and Sensor Networks (ESAS 2005*, pages 113–127, 2005.

[3] G. Acs, L. Buttyan, and I. Vajda. Provably secure on-demand source routing in mobile ad hoc networks. In *IEEE Transactions on Mobile Computing*, volume 5, 2006.

[4] G. Acs, L. Buttyan, and I. Vajda. The security proof of a link-state routing protocol for wireless sensor networks. In *IEEE Workshop on Wireless and Sensor Networks Security*, 2007.

[5] T. R. Andel and A. Yasinsac. Automated evaluation of secure route discovery in manet protocols. In *SPIN '08: Proceedings of the 15th international workshop on Model Checking Software*, pages 26–41, 2008.

[6] B. Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *IEEE Symposium on Security and Privacy*, pages 86–100, Oakland, California, May 2004.

[7] L. Buttyán and T. V. Thong. Formal verification of secure ad-hoc network routing protocols using deductive model-checking. http://www.crysys.hu/tvth/adhocTechreport2010.pdf, 2010.

[8] L. Buttyán and I. Vajda. Towards provable security for ad hoc routing protocols. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 94–105, 2004.

[9] C. Fournet and M. Abadi. Mobile values, new names, and secure communication, 2001.

[10] J. C. Godskesen. A calculus for mobile ad hoc networks. In *COORDINATION*, pages 132–150, 2007.

[11] Y.-C. Hu and A. Perrig. A survey of secure wireless ad hoc routing. *IEEE Security and Privacy*, 2(3):28–39, 2004.

[12] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, 1996.

[13] P. Papadimitratos and Z. Haas. Secure routing for mobile ad hoc networks. In *Networks and Distributed Systems Modeling and Simulation*, 2002.

[14] A. Singh, C. R. Ramakrishnan, and S. A. Smolka. A process calculus for mobile ad hoc networks. *Sci. Comput. Program.*, 75(6):440–469, 2010.