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

Ta Vinh Thong
thong@crysys.hu

Budapest University of Technology and Economics,
Department of Telecommunications,
Laboratory of Cryptography and System Security (CrySyS)

TECHNICAL REPORT



2012

*This report is the extended and revised version of the 6 pages conference paper [9]

# Contents

**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 algebra 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, and provides expressiveness for explicitly modelling cryptographic privitives.

# 1 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., [15] 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. The examples presented in this paper mainly consider secure on-demand source routing protocols, however, the general idea and methodology can be used to reasoning of other routing protocols as well. Our approach is based on a new process algebra 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 [10, 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 2 we give an overview of the SRP protocol [19] and the Ariadne protocol [16] and the attacks that have been found against them. In these attacks the attacker node creates an incorrect routing state by modifying control messages during the route discovery phase so that the incorrect route is accepted as if it is correct. In this work we focus on modelling and verifying the occurence of this kind of attacks. In Section 3, we provide the discussion on the most important related works and emphasizing the difference between them and our work. In Section 4, we give the detailed discussion of the syntax as well as the semantics of our process algebra. In Section 6, we demonstrate the expressive power of our algebra by modelling the operation of SRP and a known security flaw in SRP. In Section 6 we demonstrate the application of the *sr*-calculus by modelling the SRP protocol and an attack on it. In Section 9, we discuss how the verification process can be automated and describe our deductive proof technique. In Section 10 we demonstrate the application of our automatic verification method by detecting an attack on the SRP protocol. Finally, in Section 11, we conclude the paper and discuss our planned future work on this topic.

## 2 Some attacks against secure routing protocols

Several "secure" routing protocols have been proposed in the recent past for wireless ad hoc networks. However, later most of them are turned out to be vulnerable to various attacks. In this section, we introduce some of these attacks that serves as the motivation of our work. First, we discuss the attack called as *relay attack*, which has already been modelled in the previous works [18, 5, 12]. Afterwards we give an overview of more subtle attacks against the SRP and the Ariadne protocols [10, 3]. We emphasize that these kinds of attacks cannot

be modelled directly and conveniently in the previous works [18, 12, 23]. Our emphasis is deliberately on modelling and verifying these kinds of attacks.

## 2.1 Relay attacks

*Relay attacks* are such kind of attacks in which the attacker node forwards the received message unchaged. A typical relay attack scenario against on-demand source routing protocols is shown in the Figure 1. In this scenario the attacker node $A$ receives the request message that includes the route specified by the list of node IDs $[\ldots, N]$. At this point, node $A$ should append its own identifier to the ID list above and forwards the request containing the ID list $[\ldots, N, A]$ to the node $M$. However, instead the attacker forwards the message unchanged to $M$. Then, node $M$ appends it own identifier to the ID list and forwards it. This procedure continues until the request message reached the destination node. At this point after making verification steps the destination node sends back a reply message. The reply message propagates backward along the route in the request message. After a while the reply reaches node $M$ and is forwarded "backward". Due to the wireless environment and the fact that the attacker node is within the transmission range of node $M$, the reply is intercepted by the attacker. When the attacker node receives the reply message it forwards it unchanged. At the end, the source node accepts the route $[\ldots, N, M, \ldots]$ which does not exist in the current topology.

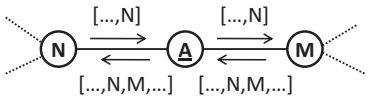


Figure 1: A typical relay attack scenario against on-demand source routing protocols.

## 2.2 An attack against the SRP protocol

SRP is a secure on-demand source routing protocol for ad-hoc networks proposed in [19]. The design of the protocol is inspired by the DSR protocol [17], 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.

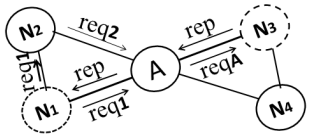


Figure 2: An attack scenario against the SRP protocol.

In order to illustrate a known attack on SRP, let us consider the network topology shown in Figure 2. 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 $(N_2, \lambda, N_4)$, where $\lambda$ is an arbitrary sequence of fake identifiers. 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 ($N_1$, $N_2$, $\lambda$, $N_4$, $N_3$). However, this is a mistake, because there is no route via nodes $N_2$, $\lambda$, $N_4$.

## 2.3 An attack against the Ariadne protocol

In this subsection, we show the attack has been found in [10] against the Ariadne secure routing protocol.

Ariadne has been proposed in [16] as a secure on-demand source routing protocol for ad hoc networks. Ariadne comes in three different flavors corresponding to three different techniques for data authentication. More specifically, authentication of routing messages in Ariadne can be based on TESLA [20], on digital signatures, or on MACs. We discuss Ariadne with digital signatures.

There are two main differences between Ariadne and SRP. First, in Ariadne not only the initiator and the target authenticate the protocol messages, but intermediate nodes too insert their own digital signatures in route requests. Second, Ariadne uses per-hop hashing to prevent removal of identifiers from the accumulated route in the route request. The initiator of the route discovery generates a route request message and broadcasts it to its neighbors. The route discovery message contains the identifiers of the initiator and the target, a randomly generated request identifier, and a MAC computed over these elements with a key shared by the initiator and the target. This MAC is hashed iteratively by each intermediate node together with its own identifier using a publicly known one-way hash function. The hash values computed in this way are called per-hop hash values. Each intermediate node that receives the request for the first time re-computes the per-hop hash value, appends its identifier to the list of identifiers accumulated in the request, and generates a digital signature on the updated request. Finally, the signature is appended to a signature list in the request, and the request is re-broadcast. When the target receives the request, it verifies the perhop hash by re-computing the initiators MAC and the perhop hash value of each intermediate node. Then it verifies all the digital signatures in the request. If all these verifications are successful, then the target 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 identifiers of the target and the initiator, the route and the list of digital signatures obtained from the request, and the digital signature of the target on all these elements. Each intermediate node passes the reply to the next node on the route (towards the initiator) without any modifications. When the initiator receives the reply, it verifies the digital signature of the target and the digital signatures of the intermediate nodes (for this it needs to reconstruct the requests that the intermediate nodes signed). If the verifications are successful, then it accepts the route returned in the reply.

Let us consider Figure 3, which illustrates part of a configuration where the discovered attack is possible. The attacker is denoted by A. Let us assume that S sends a route request towards D. The request reaches $V$ that re-broadcasts it. Thus, $A$ receives the following route request message:

$$reqV = (rreq,\ S,\ D,\ ID,\ h_V,\ (V),\ (sig_V))$$

where $ID$ is the random request identifier, $h_V$ is the per-hop hash value generated by $V$, and $sig_V$ is the signature of $V$.

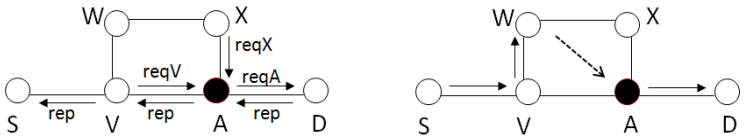


Figure 3: A subtle attack against Ariadne. The figure on the left shows the communication during the route discovery, while the figure on the right illustrates that at the end of the route discovery phase, the source node accepts the route $S, V, W, A, D$, which is not valid because the link between $W$ and $A$ does not exist.

After receiving $req_V$ the attacker waits for an another route request from $X$:

$$reqX = (rreq,\ S,\ D,\ ID,\ h_X,\ (V,W,X),(sig_V,\ sig_W,\ sig_X)).$$

From $req_X$, $A$ knows that $W$ is a neighbor of $V$. $A$ computes $h_A = H(A, H(W, h_V))$, where $h_V$ is obtained from $req_1$, and $H$ is the publicly known hash function used in the protocol. $A$ obtains the signatures $sig_V$, $sig_W$ from $req_2$. Then, $A$ generates and broadcasts the following request:

$$reqA = (rreq,\ S,\ D,\ ID,\ h_A,\ (V,W,A)\ (sig_V,\ sig_W,\ sig_A))$$

Later, $D$ generates the following route reply and sends it back towards $S$:

$$rep = (rreq,\ D,\ S,\ (V,W,A),\ (sig_V,\ sig_W,\ sig_A),\ sig_D).$$

When $A$ receives this route reply, it forwards it to $V$ in the name of $W$. Finally, $S$ will output the route $(S,V,W,\ A,\ D)$, which is a non-existent route.

### 2.4 Summary

To sum up this section, we note that the modelling of the relay attacks has been addressed in some related works [12, 18]. However, relay attacks are mainly concerned in *neighbor discovery* [21] instead of route discovery. In contrast, in our work we primarily focus on formal verification of the attacks concerning *routing*, such as the attack against the SRP and Ariadne protocols we shown above. We note that modelling the attacks against the SRP and Ariadne protocols is more difficult than modelling relay attacks because they required the modelling of more complex attackers.

## 3 Motivation and Related works

As we can see, the discussed attacks are very subtle, thus, it is hard to detect and reasoning about them manually. In addition, reasoning in the hand and pencil manner is error-prone, therefore a systematic and automatic method is required.

Our purpose in this paper is to provide a formal modelling of secure on-demand source routing protocols and a systematic and automatic method for detecting attacks similar to the attacks we introduced in the previous section. Till now, there are only few works address directly this problem. Each method proposed in the most important related works [12, 13, 23, 5, 18, 10, 3, 24, 22] has numerous drawbacks that we will discuss in the following:

In works [10, 3] the authors model the operation of the protocol participants by interactive and probabilistic Turing machines, where the interaction is realized via common tapes. This model enables us to be concerned with arbitrary feasible attacks. The security objective function is applied to the output of this model (i.e., the final state of the system) in order to decide whether the protocol functions correctly or not. Once the model is defined, the goal is to prove that for any adversary, the probability that the security objective function is not satisfied is negligible.

The main drawback of this method is that the proof is not systematic and automated, and the framework is not well-suited for detecting attack scenarios once the proof fails.

In this paper we aim at improving these works by adding automated verification method based on deductive model-checking.

In order to give a formal and precise mathematical reasoning of the operation of routing protocol for mobile ad-hoc networks several process calculi have been proposed in the recent years. Among them the two works [12, 23] are closest to our work.

In the work [12] the author proposes the process algebra that is called as CMAN for formal modelling of mobile ad-hoc networks. The advantage of this method is that it provide the modelling of cryptographic primitives and it is focused mainly on modelling mobilily nature of mobile ad-hoc networks. The drawback of this method is that it cannot be directly used for modelling such attacks as the attacks scenario against the Ariadne or SRP protocols we showed in the Section 2. In the attack scenario against the Ariadne protocol the attacker node waits, collects information it intercepts from the neighbor nodes and use them to construct an message that contains an incorrect route. CMAN does not provide syntax and semantics for modelling a knowledge base of the attacker node. In order to directly model these kind of attacks we propose the notion of the *active substitution with range* in the $sr$-calculus.

In the work [23] the authors propose the process algebra that is called as the $\omega$-calculus. The main advantage of this calculus is that it provides the direct modelling of broadcast communication and mobility. The main drawback of this method that it does not provide the syntax and semantics for modelling cryptographic primitives and attacker's knowledge base. In contrast to this our calculus provides the modelling of cryptographic primitives and attacker accumulated knowledge.

The advantage of these process algebras is that the operation of mobile ad-hoc networks and several properties such as loop-freedom and security properties can be precisely and systematically described with them, however, the drawback of them is that the proofs and reasoning are still performed manually by hand.

Several works in the literature address the problem of automatic verification of routing protocols. In the works [24, 22] the authors investigate the problem of verifying loop freedom property of routing protocols. In [24] the LUNAR protocol is verified using the SPIN, and UPPAAL model-checkers; in [24] the authors verified the DYMO protocol using graph transformation. In contrast to these works we proposed an automatic verification method focuses on verifying security properties of "secure" routing protocols instead of loop freedom property.

The two works that are most related to our work are [5, 18]. In the work [5], and [18] the authors address the problem of verification security properties of secure routing protocols using the SPIN model-checker and CPAL-ES tools, respectively.

The main drawbacks of these methods are that they suffer from expressiveness limitation. In particular, they cannot directly model cryptographic primitives and boadcast communications, instead they simulate cryptographic primites with a series of bytes [5] and broadcast communication with a sequence of unicast communication. In contrast to these works, our automatic verification method provides a direct modelling of cryptographic primitives and neighborhood.

# 4 The *sr*-calculus

In this section we define the proposed calculus: its syntax and informal semantics, as well as its operational semantics. We call this calculus as *sr*-calculus, where the prefix *sr* refers to the words **s**ecure **r**outing.

The advantage of the *sr*-calculus is that it provides expressiveness for modelling broadcast communication, neighborhood, mobility, and transmission range like in CMAN [12] and the $\omega$-calculus [23], and the explicit modelling of cryptographic primitives like in the applied $\pi$-calculus [11], however, compared to them it includes the novel definition of *active substitution with range* that enables us to model attacker knowledge and attacks in the context of ad-hoc mobile networks.

CMAN cannot be used to directly model the attacks we found against the well-known SRP and Ariadne protocols [10, 3]. In these attacks the attacker can receive information from several paths, the attacker node then collects and store these information and construct a message that contains an incorrect route.

The $\omega$-calculus lacks of modelling cryptographic primitives, such as digital signature and hashing, and has been used for modelling loop-freedom properties of AODV. Hence, it also lacks sytax and semantics for modelling attacker's knowledge set. In addition, neighbor nodes is organized into groups. However, it is not easy to determine groups in the topology.

Finally, the applied pi-calculus [11] provides active substitution that can be used to model actual attacker's knowledge. However, it lacks syntax and semantics for broadcasting, neighbors and deals with Dolev-Yao attacker model, which is not true in case of MANET.

We combine the advantage of each in order to provide a calculus with which we can directly and conveniently modelling and proving security properties and reasoning about attacks discussed in Section 2.

## 4.1 Type system of the *sr*-calculus

In this subsection we provide a basic type system for the proposed calculus. The main purpose of the type system is to reduce the number of the possible cases to be examined during the formal security proofs. Based on the type system we are capable of capturing errors such as arity mismatch and errounous binding/substitution of terms. We adopt the type system proposed for the applied $\pi$-calculus, discussed in the chapter 4 of [8], which have been shown to be sound and complete. This type system includes a syntax and a semantics part, which discuss the declaration of the types and the rules for typing, for example, the type preserved property of transitions. In this paper, we only provide a brief overview of the type system in the chapter 4 of [8].

The type system catches the errors such as arity errors and matching of terms of different type. The type system does not include recursive types, so the type of processes such as $\overline{c}\langle c\rangle.P$ is not defined.

## 4.2 Basic definitions and terminology

**Definition 1.** *Type assignment is an assignment $v : T$ (or $u : T$) of a type $T$ to $v$ (or $u$) that can be a name, a constant, a node ID or a variable.*

**Definition 2.** *A well-formed type environment $\Gamma$ is a finite set of type assignments where all names and variables are distinct. The domain of $\Gamma$ is $dom(\Gamma)$ $= \{\ v \mid \exists\ T.\{\ v : T\} \in \Gamma\ \}$*

**Definition 3.** *Let $\Gamma$ and $\Delta$ type environments. We say that $\Gamma$ extends $\Delta$ if the following holds:*

- $dom(\Gamma) \subseteq dom(\Delta)$
- *if $v \in dom(\Delta)$ such that $\{v : T\} \in \Delta$, then $\{v : T\} \in \Gamma$.*

If $\Delta = \{v_1 : T_1,\ldots,\ v_n : T_n\ \}$ and $\Gamma = \{u_1 : T_1,\ldots,\ u_n : T_n\ \}$ and $dom(\Gamma) \cap dom(\Delta) = \emptyset$, then $\Delta$ can be extended with $\Gamma$ by taking their union, denoted by $\Delta \uplus \Gamma$.

## 4.3 Simple type system for the applied $\pi$-calculus

The set of types is divided into the sets of *term types* and *process/behavior types*. Within the *term types* we distinguish among *channel types*, *broadcast types*, *name types*, *variable types*, *constant types*, and *node ID types*.

Given a term type $T_t$, channel and broadcast channel types are constructed by the unary type constructors $ch(T_t)$ and $bch(T_t)$, which are the types that is allowed to carry data with term type.

Let $\Gamma$ be a type environment and $\Lambda$ an expression which may be either a term, a process, or an extended process. A type judgment $\Gamma \vdash \Lambda : T$ is an assertion that the expression $\Lambda$ has type $T$ under the assumptions given in $\Gamma$. In particular, $\Gamma \vdash \Lambda : T$, asserts the following depending on $\Lambda$. If $\Lambda$ is a term, then $T$ is term type $T_t$. Thus $\Gamma \vdash t : T_t$ asserts that $t$ has term type under the assumptions of $\Gamma$. The so called behavior type, denoted by $T_{proc}$, is introduced for processes. $\Gamma \vdash P : T_{proc}$ asserts that process $P$ respects the type assertions in $\Gamma$. The judgment ($\vdash \Gamma$ *well-formed*) means that $\Gamma$ is a well-formed type environment.

The types for the *sr*-calculus are generated by the grammar:

$$
\begin{array}{rll}
S, T ::= & T_t \mid T_{proc} & (\textbf{Types}) \\
T_t ::= & T_{ch} \mid T_{br} \mid T_{str} & (\text{Term Types}) \\
T_{str} ::= & T_{name} \mid T_{var} \mid T_f \mid T_{const} & (\text{String Types}) \\
T_{name} ::= & tn_1 \mid \ldots \mid tn_n & (\text{Name Types}) \\
T_{var} ::= & tv_1 \mid \ldots \mid tv_n & (\text{Variable Types}) \\
T_f ::= & f(T^1_{str}, \ldots, T^n_{str}) & (\text{Function Types}) \\
T_{const} ::= & T_{req/rep} \mid tconst_i & (\text{Constant Types}) \\
T_{ch} ::= & \mathrm{ch}(T^1_{str}, \ldots, T^n_{str}) & (\text{Channel Types}) \\
T_{br} ::= & \mathrm{bch}(T^1_{str}, \ldots, T^n_{str}) & (\text{Broadcast Channel Types}) \\
T_{id} ::= & tl_1 \mid \ldots \mid tl_n & (\text{Node ID Types}) \\
T_{proc} ::= & tp_1 \mid \ldots \mid tp_n & (\text{Process/Behavior Types})
\end{array}
$$

where $tn$, $tv$, $tl$ and $tp$ are name, variable, node ID, and process types, respectively. The abbreviation of $x_1 : T_1, \ldots, x_n : T_n$ is defined by $\vec{x} : \vec{T}$. Of course, if a term $t$ has a string type $T_{str}$ then it also has a term type $T_t$, and if $t$ has been assigned to one of the type $T_{name}$, $T_{var}$, $T_f$, $T_{const}$ then it implicitly has a type $T_{str}$. The reverse direction is not always true, hence, to avoid type conflict the most narrow type should be assigned in the declaration. Note that within the set of term type the channel types are distinguished from the remaining string types because to reasoning about routing protocols we do not need to send channels, or need not to define a function that includes channel arguments. Within the constant type we define $T_{req/rep}$ as the type for the special constants rreq and rrep which are the parts of the routing messages.

Within a function types we distinguish types of each crypto function, such as, digital signature type, $T_{sig}$, one-way hash type, $T_{hash}$, MAC function type, $T_{mac}$. We also define types of secret key, $T_{skey}$, public key $T_{pkey}$, and symmetric shared key $T_{shkey}$. In this paper we only use these three crypto functions, but of course any function types can be defined in the similar way. With these types we can ease the security verification, and reducing the number of possibilities.

$$
\begin{array}{rll}
T_{skey} ::= & sk(T_{id}) & (\text{Secret Key Types}) \\
T_{pkey} ::= & pk(T_{id}) & (\text{Public Key Types}) \\
T_{shkey} ::= & k(T_{id}, T_{id}) & (\text{Shared Key Types}) \\
T_{sig} ::= & sign(T_{str}, T_{skey}) & (\text{Digital Signature Types})
\end{array}
$$

$T_{hash} ::= hash(T_{str})$ (One-Way Hash Types)
$T_{mac} ::= mac(T_{str}, T_{key})$ (MAC Types)

The syntax, reduction rules and trasition rules for the typed applied $\pi$-calculus remains unchanged from the one for the untyped applied $\pi$-calculus.

In order to ensure that structural equivalence preserves well-typedness, we require that the type system assigns equal types to terms that are equated by the equational theory.

**Definition 4.** *(Well formed environment)*

- $\vdash \emptyset$ *well-formed*
- *If* ($\vdash \Gamma$ *well-formed*) *and* $u \notin dom(\Gamma)$*, then* ($\vdash \Gamma \uplus u$ *well-formed*)*.*

**Definition 5.** *(Type rules for terms) Let* $t \in \mathcal{T}$ *be a term,* $T$ *a type, and* $\Gamma$ *a well-formed type environment. Then the type judgment* $\Gamma \vdash_{\mathcal{T}} t : T$ *holds if it can be derived by application of one of the following rules.*

- *If* $\vdash \Gamma$ *well-formed and* $u : T \in \Gamma$ *then* $\Gamma \vdash_{\mathcal{T}} u : T_{str}$
- *If* $\Gamma \vdash_{\mathcal{T}} t_1 : T^1_{str} \ldots \Gamma \vdash_{\mathcal{T}} t_n : T^n_{str}$*, and the arity of* $f$ *is* $n$*, then* $\Gamma \vdash_{\mathcal{T}} f(T^1_{str}, \ldots, T^n_{str}) : T_{str}$*, for each function name* $f$*.*

**Definition 6.** *(Type rules for processes) Let* $P \in \mathcal{P}$ *be a process,* $\Gamma$ *a well-formed type environment, and* $T_{proc}$ *the behavior type. Then the judgment* $\Gamma \vdash_{\mathcal{P}} P : T_{proc}$ *holds if it can be derived by application of one of the following rules.*

- *If* ($\vdash \Gamma$ *well-formed*) *then* ($\Gamma \vdash_{\mathcal{P}} nil : T_{proc}$ )
- *If* ($\Gamma \vdash_{\mathcal{P}} P : T_{proc}$ ) *and* ($\Gamma \vdash_{\mathcal{P}} Q : T_{proc}$ )*, then* $\Gamma \vdash_{\mathcal{T}} P \mid Q : T_{proc}$*.*
- *If* ($\Gamma \vdash_{\mathcal{P}} P : T_{proc}$ ) *then If* ($\Gamma \vdash_{\mathcal{P}} !P : T_{proc}$ )
- *If* ($\Gamma \uplus \{u : T_{str}\} \vdash_{\mathcal{P}} P : T_{proc}$ ) *then* ($\Gamma \vdash_{\mathcal{P}} \nu u.P : T_{proc}$ )
- *If* ($\Gamma \vdash_{\mathcal{T}} t_1 : T_{str}$)*,* ($\Gamma \vdash_{\mathcal{T}} t_2 : T_{str}$)*,* ($\Gamma \vdash_{\mathcal{P}} P : T_{proc}$ )*,* ($\Gamma \vdash_{\mathcal{P}} Q : T_{proc}$ ) *then* ($\Gamma \vdash_{\mathcal{P}} [t_1 = t_2]\ P$ *else* $Q : T_{proc}$ )*, and also* ($\Gamma \vdash_{\mathcal{P}} [t_1 = t_2]\ P : T_{proc}$ ).
- *If* ($\Gamma \vdash_{\mathcal{T}} l : T_{id}$)*,* ($\Gamma \vdash_{\mathcal{T}} \sigma : T_{id}$)*,* ($\Gamma \vdash_{\mathcal{P}} P : T_{proc}$ )*,* ($\Gamma \vdash_{\mathcal{P}} Q : T_{proc}$ ) *then* ($\Gamma \vdash_{\mathcal{P}} [l \in \sigma]\ P$ *else* $Q : T_{proc}$ )*, and also* ($\Gamma \vdash_{\mathcal{P}} [l \in \sigma]\ P : T_{proc}$ ).
- *If* ($\Gamma \uplus \{\vec{x} : \vec{T}_{str}\} \vdash_{\mathcal{P}} P : T_{proc}$ ) *and* ($\Gamma \vdash_{\mathcal{T}} c : ch(\vec{T}_{str})$) *then* ($\Gamma \vdash_{\mathcal{P}} c(\vec{x}).P : T_{proc}$)*.*
- *If* ($\Gamma \vdash_{\mathcal{T}} \vec{t} : \vec{T}_{str}$) *and* ($\Gamma \vdash_{\mathcal{T}} c : ch(\vec{T}_{str})$) *and* ($\Gamma \vdash_{\mathcal{P}} P : T_{proc}$ ) *then* ($\Gamma \vdash_{\mathcal{P}} \overline{c}\langle \vec{t}\, \rangle.P : T_{proc}$ ).
- *If* ($\Gamma \uplus \{\vec{x} : \vec{T}_{str}\} \vdash_{\mathcal{P}} P : T_{proc}$ ) *and* ($\Gamma \vdash_{\mathcal{T}} br : bch(\vec{T}_{str})$) *then* ($\Gamma \vdash_{\mathcal{P}} br(\vec{x}).P : bch(\vec{T}_{str})$)*.*

- *If* $(\Gamma \vdash_{\mathcal{T}} \vec{t} : \vec{T}_{str})$ *and* $(\Gamma \vdash_{\mathcal{T}} br : bch(\vec{T}_{str}))$ *and* $(\Gamma \vdash_{\mathcal{P}} P : T_{proc})$ *then* $(\Gamma \vdash_{\mathcal{P}} \overline{br}\langle\vec{t}\rangle.P : T_{proc})$. *In the rest of the paper to represent broadcast sending/receiving we simply use* $\langle\vec{t}\rangle.P$, *and* $(x).P$ *by removing br from the beginning.*

- *If* $(\Gamma \vdash_{\mathcal{T}} t : T_{str})$ *and* $(\Gamma \vdash_{\mathcal{T}} x : T_{str})$ *then* $(\Gamma \vdash_{\mathcal{P}} \{t/x\} : T_{proc})$.

## 4.4 Syntax and informal semantics of the *sr*-calculus

We assume an infinite set of *names* $\mathcal{N}$ and *variables* $\mathcal{V}$, where $\mathcal{N} \cap \mathcal{V} = \emptyset$. Further, we define a set of *node identifiers* (node ID) $\mathcal{L}$, where $\mathcal{N} \cap \mathcal{L} = \emptyset$. Each node identifier uniquely identifies a node, that is, no any node pair shares the same identifier.

We define a set of terms as

$$t ::= req \mid rep \mid true \mid ACCEPT \mid c, n, m, k \mid l_i, l_a \mid x, y, z, v, w \mid f(t_1, \ldots, t_k)$$

- *req* and *rep* are unique constants that represent the *req* and *rep* tags in route request and reply messages;

- *true* is a special constant that models the logical value 1;

- *ACCEPT* is a special constant. The source node outputs *ACCEPT* when it receives the reply message and all the verifications it makes on the reply are successful.

- $c$ models communication channel;

- $n$, $m$ and $k$ are names and are used to model some data (e.g., a random nonce, a secret key);

- $l_i$, $i$ can be numbers or letters not equal to $a$, and $l_a$ are node IDs of the honest and the attacker node, respectively;

- $x$, $y$, $z$, $v$ and $w$ are variables that can represent any term, that is, any term can be bound to variables.

- Finally, $f$ is a constructor function with arity $k$ and is used to construct terms and to model cryptographic primitives, 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 $(t_1,\ldots,t_k)$.

*We note that, this definition of term is new compared to CMAN, ω-calculus, and the applied pi-cacluclus in that it includes constants rep, req for modelling route discovery protocols, and process* $[l \in \sigma]P$ *for examining the neighborhood.*

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

| | |
|---|---|
| $P, Q, R ::=$ | processes |
| $\overline{c}\langle t\rangle.P$ | unicast send |
| $c(x).P$ | unicast receive |
| $\langle t\rangle.P$ | broadcast send |
| $(x).P$ | broadcast receive |
| $P\|Q$ | composition |
| $\nu n.P$ | restriction |
| $!P$ | replication |
| $[t_i = t_j]P$ | if |
| $[l \in \sigma]P$ | in |
| $\mathbf{0}$ | nil |
| $let\ (x = t)\ in\ P$ | let |

*We note that, this definition of processes is novel compared to $\omega$-calculus in that it includes constructor and destructor applications. Constructor/destructor application is used to model cryptographic primitives. Compared to CMAN it includes also the posibility of unicast. Finally, it differs from the applied pi-calculus in that it includes broadcast send and receive actions.*

- The processes $\overline{c}\langle t\rangle.P$ represents the sending of message $t$ on channel $c$ followed by the execution of $P$, and $c(x).P$ represents the receiving of some message and binds it to $x$ in $P$. For example, the communication between $\overline{c}\langle t\rangle.P$ and $c(x).P$ can be described as the reduction step of the parallel composition, namely, $\overline{c}\langle t\rangle.P \mid c(x).P \longrightarrow P \mid P\{t/x\}$. These two process model the unicast receive actions.

- The two processes $\langle t\rangle.P$, and $(x).P$ represent the broadcast send and receive. Compared to the unicast case they does not contain the channel, which intends to model the fact that there is no any specified target.

- A composition $P|Q$ behaves as processes $P$ and $Q$ running in parallel. Each may interact with the other on channels known to both, or with the outside world, independently of the other. Given a family of processes $P_1, P_2, \ldots P_k$, we write $\prod_{i\in 1\ldots k} P_i$, or $\prod_{i\in\{1\ldots k\}} P_i$ for their parallel composition $P_1|P_2|\ldots|P_k$.

- A restriction $\nu n.P$ is a process that makes a new, private name $n$, and then behaves as $P$.

- A replication $!P$ behaves as an infinite number of copies of $P$ running in parallel.

- Processes $[t_i = t_j]P$ and $[l \in \sigma]P$ mean that if $t_1 = t_2$ and $l \in \sigma$, respectively, then process $P$ is "activated", else they get stuck and stay idle.

- The *nil* process $\mathbf{0}$ does nothing.

- Finally, *let* $(x = t)$ *in* $P$ means that the procedure of binding $t$ to free occurrences of $x$ in process $P$.

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

A *networks*, denoted as $N$, can be an empty network with no nodes: $0_N$; a singleton network with one node: $\lfloor P \rfloor_l$; the parallel composition of nodes: $\lfloor P_1 \rfloor_{l_1}^{\sigma_1} \mid \lfloor P_2 \rfloor_{l_2}^{\sigma_2}$, where $\sigma_1$ and $\sigma_2$ may include $l_2$ and $l_1$ respectively; a network with name restriction, and the parallel composition of networks: $N_1 \mid N_2$.

$$N ::= \mathbf{0}_N \mid \lfloor P \rfloor_l \mid (\lfloor P_1 \rfloor_{l_1}^{\sigma_1} \mid \lfloor P_2 \rfloor_{l_2}^{\sigma_2}) \mid \nu n.N \mid (N_1 \mid N_2)$$

Note that, $\lfloor P \rfloor_{l_1}^{l_2}$ and $\lfloor Q \rfloor_{l_2}^{l_1}$ means the node $l_1$ and node $l_2$ are bidirectionally connected.

*We stress that we use the form* $\lfloor P \rfloor_l^\sigma$ *that is already proposed in CMAN, and not the definition of groups (where neighbor nodes is organized to the same group) as in* $\omega$*-calculus because we found that the topology of Mobile Ad-hoc Networks is usually represented as graph and given in an adjacency matrix, thus, using the groups method in* $\omega$*-calculus an algorithm can be required which extracts the cliques in the graph. Moreover, there can be redundant groups that should be handle properly for efficiency purpose. Finally, we prefer the* $\lfloor P \rfloor_l^\sigma$ *form because it gives us a possibility to model uni-directional links, which is not the case in* $\omega$*-calculus. However, we note that CMAN does not include syntax and semantics for uni-directional links.*

$\nu n.N$ represents the creation of new name $n$, such as secret keys, a nonce and only $N$ knows it.

In order to modelling attacker's knowledge base and make modelling the attacks, where attacker waits, collects and stores information feasible and more convenient, we extend the definition of networks with *the substitution with range* and the restriction on variables. Additionally, we adapt the the notion of active substitution for modelling the attacker's actual knowledge.

Again, we emphasize that the notion of active substitution and static equivalence have been used in the applied pi-calculus, however, it models the knowledge of a Dolev-Yao attacker who eavesdrops every message that has been sent by communicating partners without considering the attacker model in wireless ad-hoc networks.

*Our contribution results in slightly modifying the notion active substitution to model such an adversary who can only intercept messages sent by its neighbors. More precisely, the intercepting of broadcasted information is restricted to only nodes in the broadcast range. Furthermore, we adapt the concept of active substitution for modelling the attacker's knowledge set, which can continuesly change during the protocol. Finally, we emphasize that this is novel compared to all of the three calculi: CMAN,* $\omega$*-calculus, and applied pi-calculus.*

The definition of the *extended network* is as follows:

| | |
|---|---|
| $E ::=$ | extended network |
| $N$ | plain network |
| $E_i \mid E_j$ | parallel composition |
| $\nu n.E$ | name restriction |
| $\nu x.E$ | variable restriction |
| $\lfloor \{t/x\}^\sigma \rfloor$ | active substitution with range |

- $N$ is a plain network we already discussed above.
- $E_i|E_j$ is a parallel composition of two extended networks.
- $\nu n.E$ is a restriction of the name $n$ to $E$.
- $\nu x.E$ is a restriction of the variable $x$ to $E$.
- $\lfloor \{t/x\}^\sigma \rfloor$, which is abbreviated as $\{t/x\}^\sigma$ in the rest of the paper: $\{t/x\}^\sigma$ means that the substitution $\{t/x\}$ is applied to any node that is in parallel composition with $\{t/x\}$ and its identifier is in the set $\sigma$. Intuitively, we can say that $\sigma$ is the range of the substitution $\{t/x\}$. Formally, we can explain the notion of active substitution with range by $\nu x.\left(\{t/x\}^\sigma \mid \prod_{l_i \in \sigma} \lfloor Q_i \rfloor_{l_i}^{\sigma_i}\right)$. This formula in turn can be defined as $\nu x.\left(\{t/x\} \mid A^\sigma\right)$, where $A^\sigma$ is the extended process (the notion of extended process $A$ is presented in the applied $\pi$-calculus) that only includes the internal behavior of nodes in $\sigma$, and $\{t/x\}$ is the active substitution known in the applied $\pi$-calculus.

We write $fv(E)$, $bv(E)$, $fn(E)$, and $bn(E)$ for the sets of free and bound variables and free and bound names of $E$, respectively. These sets are defined as follow:

$$fv(\{t/x\}^\sigma) \stackrel{def}{=} fv(t) \cup \{x\},\ fn(\{t/x\}^\sigma) \stackrel{def}{=} fn(t) \cup \{node\ IDs \in \sigma\}$$

$$bv(\{t/x\}^\sigma) \stackrel{def}{=} \emptyset,\ bn(\{t/x\}^\sigma) \stackrel{def}{=} bn(t)$$

An extended network is closed when every variable is either bound or defined by an active substitution with range.

In the applied $\pi$-calculus, a frame is an extended process built up from 0 and active substitutions of the form $\{t/x\}$ by parallel composition and restriction. Analogously, we follow this concept and we let the frame of network, denoted by $\varphi_N$, be an extended network built up from $0_N$ and active substitutions with range, $\lfloor \{t/x\}^\sigma \rfloor$ (or simply $\{t/x\}^\sigma$). Every extended network $E$ can be mapped to a frame $\varphi_N(E)$ by replacing every plain network embedded in $E$ with empty network $\mathbf{0}_N$. We distinguish the notations frame of network ($\varphi_N$) and frame of processes ($\varphi$) also known in case of the applied $\pi$-calculus.

For example, the frame of process $E$, where

$$E = \{t_1/x_1\}^{\sigma_1} \mid \{t_2/x_2\}^{\sigma_2} \mid \ldots \mid \{t_k/x_k\}^{\sigma_k} \mid \prod_i \lfloor Q_i \rfloor_{l_i}^{\sigma_i}$$

is $\varphi_N(E) = \{t_1/x_1\}^{\sigma_1} \mid \{t_2/x_2\}^{\sigma_2} \mid \ldots \mid \{t_k/x_k\}^{\sigma_k}$.

The frame $\varphi_N(E)$ can be viewed as an approximation of the behavior of $E$ that accounts for the *static knowledge* exposed by $E$ to its environment, but not for $E$'s dynamic behavior.

In the next section we give the semantics of the calculus in order to reason about secure on-demand source routing protocols.

## 4.5 Semantics

First we define the strutural equivalence relation which is used to simplify a process of large size to a smaller one that is the is equivalent to the original one. This relation is very important in proofs. We say that two processes are structurally equivelence, if they are identical up to structure.

### 4.5.1 The Structural Equivalence($\equiv$)

In particular, structural equivalence relation is defined as the least equivalence relation satisfying bound name, bound variable conversion (also called as $\alpha$-conversion) and the following rules:

**(Rules for Processes:)**

| | |
|---|---|
| (Struct P-$\alpha$) | $P \equiv_{x \leftarrow y} Q$; $P \equiv_{n_1 \leftarrow n_2} Q$ |
| (Struct P-Par1) | $P \mid \mathbf{0} \equiv P$ |
| (Struct P-Par2) | $P_1 \mid P_2 \equiv P_2 \mid P_1$ |
| (Struct P-Par3) | $(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3)$ |
| (Struct P-Switch) | $\nu n_1.\nu n_2.P \equiv \nu n_2.\nu n_1.P$ |
| (Struct P-Repl) | $!P \equiv P \mid !P$ |
| (Struct P-Drop) | $\nu n.0 \equiv 0$ |
| (Struct P-Extr) | $\nu n.(P \mid Q) \equiv P \mid \nu n.Q$ if $n \notin fn(P)$ |
| (Struct P-Let) | $let\ x = t\ in\ P \equiv P\{t/x\}$ |
| (Struct P-If1) | $[t = t]P \equiv P$ |
| (Struct P-If2) | $[t_i = t_j]P \equiv 0\ (if\ t_i \neq t_j)$ |
| (Struct P-In1) | $[l \in \sigma]P \equiv P\ (if\ l \in \sigma)$ |
| (Struct P-In2) | $[l \in \sigma]P \equiv 0\ (if\ l$ is not in $\sigma)$ |

The meaning of each rule is the following:

- Struct P-$\alpha$: P and Q are stuctural equivalent if Q can be obtained from P by renaming one or more bound names/variables in P, or vice versa. For instance, processes $(x).P$ and $(y).P$ are structural equivalent by renaming $y$ to $x$. This is denoted by $\equiv_{x \leftarrow y}$.
- Struct P-Par1: The parallel composition with the nil process does not change anything, the result is the same as the original parallel composition.
- Struct P-Par2: The parallel composition is commutative.

- Struct P-Par3: The parallel composition is associative.
- Struct P-Switch: The restriction is commutative.
- Struct P-Drop: Restriction does not affect the nil process, thus, we can drop it.
- Struct P-Extrusion: We can drop the restriction from process $P$ when $P$ does not contain the restricted name as free name, that is, the restricted name does not occur in $P$.
- Struct P-Let: Both sides represent the binding of the term $t$ to variable $x$ in P.
- Struct P-If1, P-If2: if the two terms are the same then the execution of $P$ begins, while if they are distinct then the process gets stuck.
- Struct P-In1, P-In2: If the node identifier $l$ is in the set $\sigma$ then the execution of $P$ begins, otherwise the process $P$ gets stuck and stays idle.

The next additional rules are valid to structural equivalence:

$$\frac{P \equiv Q, Q \equiv R}{P \equiv R} \qquad \frac{P \equiv P'}{P|Q \equiv P'|Q} \qquad \frac{P \equiv P'}{\nu n.P \equiv \nu n.P'}$$

The first one means structural equivelence relation is transitive: if $P \equiv Q$ and $Q \equiv R$ then $P \equiv R$; the second and third rules show that structural equivalence closed to replication and restriction. Similarly the rules for network can be defined:

**(Rules for Networks:)**
(Struct N-Par1) $N|\mathbf{0}_N \equiv N$
(Struct N-Par2) $N_1|N_2 \equiv N_2|N_1$
(Struct N-Par3) $(N_1|N_2)|N_3 \equiv N_1|(N_2|N_3)$
(Struct N-Switch) $\nu n_1 n_2.N \equiv \nu n_2 n_1.N$
(Struct N-Extr) $(\nu n.N_1)|N_2 \equiv \nu n.(N_1|N_2)$ $(if\ n \notin fn(N_2) \cup id(N_1))$
(Struct N-Node) $\lfloor P \rfloor_l^\sigma \equiv \lfloor Q \rfloor_l^\sigma$ $(if\ P \equiv Q)$
(Struct N-Rest) $\lfloor \nu n.P \rfloor_l^\sigma \equiv \nu n.\lfloor P \rfloor_l^\sigma$

$fn(N)$, $id(N)$ represents the set of free names of $N$, the set of free varibles of $N$, and the set of node identifiers in $N$, respectively. The first five rules are standard, the only rules require some words to mention is the (Struct N-Node) and (Struct N-Rest). The first means that two networks are strutural equivalent if it contains nodes with the same internal operation and they have the same identifier with same neighbors. The second rule says that a name restriction on process can be seen as a restriction on a node.

Finally the rules for the extended network $E$ are defined as follows:

*Again, we emphasize that this part is new compared to CMAN, and the $\omega$-calculus in that it enables us to model the knowledge base of the attacker node.*

*The knowledge of the attacker can improve after series of communication steps. Also it is novel compared to the applied $\pi$-calculus in that active substitution has range for modelling neighborhood.*

**(Rules for Extended Networks:)**
(Struct E-Par1) $E|\mathbf{0}_N \equiv E$
(Struct E-Par2) $E_1|E_2 \equiv E_2|E_1$
(Struct E-Par3) $(E_1|E_2)|E_3 \equiv E_1|(E_2|E_3)$
(Struct E-Extr) $(\nu n.E_1)|E_2 \equiv \nu n.(E_1|E_2)$ $(if\ n \notin fn(E_2) \cup fv(E_2) \cup id(E_1))$
(Struct E-Switch) $\nu n_1 n_2.E \equiv \nu n_2 n_1.E$
(Struct E-Intro) $\nu x.\{t/x\}^\sigma \equiv \mathbf{0}_N$
(Struct E-Try) $\{t/x\}^\sigma|E \equiv \{t/x\}^\sigma|E\{t/x\}^\sigma$
(Struct E-Rewrite) $\{t_1/x\}^\sigma \equiv \{t_2/x\}^\sigma$ (if $t_1 = t_2$)

$fn(E)$, $fv(E)$ and $id(E)$ represents the set of free names of $E$, the set of free varibles of $E$, and the set of node identifiers in $E$, respectively. The first five rules is similar and come straightforward from the rules on networks and processes. Rule (Intro) is used to introduce any active substitutions. The rule (Struct E-Rewrite) say that two active subtitutions with the same range $\sigma$, and terms are stucturally equivalent. Rule (Try) represents the trying to apply substitution $\{t/x\}^\sigma$ to the extended network $E$: For example, let $E$ be

$$E = \nu\tilde{n}(\{t_1/x_1\}^{\sigma_1} |\ldots| \{t_k/x_k\}^{\sigma_k} \mid \lfloor Q_i \rfloor_{l_i}^{\sigma_i} |\ldots| \lfloor Q_j \rfloor_{l_j}^{\sigma_j})$$

where $\tilde{n}$ is a collection of non-duplicated names. Then $E\{t/x\}^\sigma$, where $\tilde{n} \notin fn(t) \cup fv(t) \cup id(E)$, is

$$\nu x.\nu\tilde{n}.(\{t_1/x_1\}^{\sigma_1} |\ldots| \{t_k/x_k\}^{\sigma_k} \mid \lfloor Q_i \rfloor_{l_i}^{\sigma_i} \{t/x\}^\sigma \mid \ldots \mid \lfloor Q_j \rfloor_{l_j}^{\sigma_j} \{t/x\}^\sigma)$$

Intuitively, this means that the substitution is applied on every plain network. However, this substitution successes at $\lfloor Q_i \rfloor_{l_i}^{\sigma_i}$ only in case location $l_i \in \sigma$, otherwise, it has no effect. This is formally defined by the rules (E-Try) and (E-Subst):

(Struct E-Try-1) $\lfloor Q_i \rfloor_{l_i}^{\sigma_i} \{t/x\}^\sigma \equiv_{l_i \in \sigma} \lfloor Q_i\{t/x\} \rfloor_{l_i}^{\sigma_i}$
(Struct E-Try-2) $\lfloor Q_i \rfloor_{l_i}^{\sigma_i} \{t/x\}^\sigma \equiv_{l_i \notin \sigma} \lfloor Q_i \rfloor_{l_i}^{\sigma_i}$
(Struct E-Subst-1) $\{t/x\}^\sigma|\lfloor Q_i \rfloor_{l_i}^{\sigma_i} \equiv_{l_i \in \sigma} \{t/x\}^\sigma|\lfloor Q_i\{t/x\} \rfloor_{l_i}^{\sigma_i}$
(Struct E-Subst-2) $\{t/x\}^\sigma|\lfloor Q_i \rfloor_{l_i}^{\sigma_i} \equiv_{l_i \notin \sigma} \{t/x\}^\sigma|\lfloor Q_i \rfloor_{l_i}^{\sigma_i}$

In the next subsection we introdude the reduction relation that are used to model the internal operation/computation of nodes, and to model a reduction step in case of networks.

### 4.5.2 Reduction relation ($\rightarrow$)

**(Internal reduction rules for processes:)**
(Red P-Let) $let\ x = t\ in\ P \rightarrow P\{t/x\}$

(Red P-If1) $[t = t]P \ \rightarrow \ P$
(Red P-If2) $[t_i = t_j]P \rightarrow \ 0 \ (if \ t_i \neq t_j)$
(Red P-In1) $[l \in \sigma]P \ \rightarrow \ P \ (if \ l \in \sigma)$
(Red P-In2) $[l \in \sigma]P \ \rightarrow \ 0 \ (if \ l$ is not in $\sigma)$

The operations (i) binding a variable to a term in a process; (ii) checking the equality of two terms; (iii) checking the presence of a node identifier in a set of node identifier; and (iv) destructor computations such as checking digital signatures, are all internal operations of nodes. Next we introduce the internal reduction steps in a network. Internal or silent steps that can be performed by nodes are connecting and disconnecting that concern the mobility:

**(Reduction relations for mobility:)**
(Red Connect) $\lfloor P \rfloor_{l_1}^{\sigma_1} \mid \lfloor Q \rfloor_{l_2}^{\sigma_2} \rightarrow_{\{l_1 \bullet \ l_2\}} \lfloor P \rfloor_{l_1}^{\sigma_1 l_2} \mid \lfloor Q \rfloor_{l_2}^{\sigma_2}$, where $l_2$ is not in $\sigma_1$.
(Red Disconnect) $\lfloor P \rfloor_{l_1}^{\sigma_1 l_2} \mid \lfloor Q \rfloor_{l_2}^{\sigma_2} \rightarrow_{\{l_1 \circ \ l_2\}} \lfloor P \rfloor_{l_1}^{\sigma_1} \mid \lfloor Q \rfloor_{l_2}^{\sigma_2}$, where $l_2$ is not in $\sigma_1$.

The Reduction relation (Red Connect) model the scenario in which the node $l_2$ gets into the transmission range of the node $l_1$. This reduction relation is denoted as $\rightarrow_{\{l_1 \bullet \ l_2\}}$. Its counterpart is the reduction relation (Red Disconnect) is denoted as $\rightarrow_{\{l_1 \circ \ l_2\}}$ and says that node $l_2$ get out of the transmission range of the node $l_1$.

*We note that although we have defined the rules for mobility, we will not use them in our proofs because we are only considering the analysis of the attacks discussed in Secition 2. Hence, we always assume that* ***the topology does not change*** *during the attack.*

In order to reason about secure on-demand source routing protocols, and describe the operation semantics of the *sr*-calculus we introduce the labeled transition system of the calculus in the following subsection. Labelled transition system is very important in proofs, and captures such activities made by nodes that can be observed by the environment. This activity , for instance, is broadcast sending.

### 4.5.3 Labeled transition system ($\xrightarrow{\alpha}$)

We note that broadcast and unicast sending are non-deterministically executed, while receiving actions can take place at the same time.

The traditional operation semantics for processes is defined as a labeled transition system $(\mathcal{P}, \mathcal{G}, \rightarrow)$ where $\mathcal{P}$ represents a set of processes, $\mathcal{G}$ is a set of labels, and $\rightarrow \subseteq \mathcal{P} \times \mathcal{G} \times \mathcal{P}$. In our case it is the ternary relation $(\mathcal{N}, \mathcal{G}, \rightarrow)$ where $\mathcal{N}$ represents a set of extended networks, $\mathcal{G}$ is a set of labels, and $\rightarrow \subseteq \mathcal{N} \times \mathcal{G} \times \mathcal{N}$. The following labeled transitions are specified in this transition system in the *sr*-calculus.

**(Labeled transition rules for networks)**
(Ext BroadSend1) $\nu\tilde{n}.\lfloor \langle t \rangle.P \rfloor_l^{\sigma} \ \xrightarrow{\nu x.\langle x \rangle : \bar{l}\{\sigma\}} \ \nu\tilde{n}.(\{t/x\}^{\sigma} \mid \lfloor P \rfloor_l^{\sigma})$
(Ext BroadSend2) $\lfloor \langle t \rangle.P \rfloor_l^{\sigma} \ \xrightarrow{\nu x.\langle x \rangle : \bar{l}\sigma} \ (\{t/x\}^{\sigma} \mid \lfloor P \rfloor_l^{\sigma})$

(Ext BroadRecv1) $\lfloor (x).Q \rfloor_l^{\sigma_2} \xrightarrow{(t)^{\sigma}:\{l \in \sigma\}} \lfloor Q\{t/x\} \rfloor_l^{\sigma_2}$

(Ext BroadRecv2) $\nu\widetilde{n}.\lfloor (x).Q \rfloor_l^{\sigma_2} \xrightarrow{(t)^{\sigma}:\{l \in \sigma\}} \nu\widetilde{n}.\lfloor Q\{t/x\} \rfloor_l^{\sigma_2}$

New names $\tilde{n}$ typically represent some secret key or nonce in secure on-demand source routing protocols. The rules (Ext BroadSend1-2) say that the node $l$ has broadcast term $t$, hence, it is now available for nodes in its range, $\sigma$. This is modelled by $\{t/x\}^{\sigma}$ and $\nu x$, which restricts the substitution to nodes within the range. The rules (Ext BroadRecv1-2) say that if the listening node $l$ is within $\sigma$ (which is the range of the node that sent $t$, denoted by $(t)^{\sigma}$) then it obtains $t$.

## 4.6 Equational Theory

The destructor applications (e.g., proposed in the spi-calculus), which basically is the inverse of functions, and are used to model verification computations on messages. Formally, it is the process *let* $(x = g(t_1, \ldots, t_n))$ *in* $P$ *else* $Q$, which tries to evaluate $g(t_1, \ldots, t_n)$ if this succeeds, $x$ is bound to the result and $P$ is executed, otherwise, $Q$ is executed. For instance, a typical destructor can be verification of digital signature as $checksign(sign(x, sk(y)), pk(sk(y)))$, where the constructor $pk(sk(y))$ represents the public key generated from the given secret key.

To make the proofs and the system specification be more simpler, instead of using destructor applications, we use the notion of equational theory proposed in the applied $\pi$-calculus. An equational theory $Eq$ is defined over the set of function symbols $\sum$. It contains a set of equations of the form $t_1 = t_2$, where terms $t_1$, $t_2$ are defined in $\sum$. Like the destructor application it allows us to capture relationships between terms defined in $\sum$. Equality modulo the equational theory, written $=_{Eq}$, is defined as the smallest equivalence relation on terms, that contains $Eq$ and is closed under application of function symbols, substitution of terms for variables and bijective renaming of names [11]. For instance, *dec(enc(x, y),y) = x* and *checksign(sign(x, y),pk(y)) = true* for a special constant *true*. Note that we write $=$ instead of $=_{Eq}$ for simplicity because in our case it is clear from the context.

## 4.7 Examples

Next we show the application of active substitution with range and the defined labeled transition system on two example networks. The first example network illustrated in the Figure 4 includes three nodes.

### 4.7.1 Example for broadcasting and message loss

In this simple network node $P$ is assigned the identifier $l_1$ node $Q$ has identifier $l_2$ and node $R$ is at $l_3$. Node $P$ and node $Q$ are neighbors, but node $P$ and $R$ are not. Thus, when $P$ broadcasts message $t$, only $Q$ receives $t$. The followong labeled transitions model the procedure in which $P$ broadcast $t$, and only $Q$ received this.


Figure 4: An example network

$$\Big(\lfloor\langle t\rangle.P_1\rfloor_{l_1}^{l_2} \mid \lfloor(y).Q_1\rfloor_{l_2}^{l_1} \mid \lfloor(z).R_1\rfloor_{l_3}\Big) \xrightarrow{\nu x.\langle x\rangle:\overline{l_1}\{l_2\}}$$
$$\Big(\{t/x\}^{\{l_1,l_2\}} \mid \lfloor P_1\rfloor_{l_1}^{l_2} \mid \lfloor(y).Q_1\rfloor_{l_2}^{l_1} \mid \lfloor(z).R_1\rfloor_{l_3}\Big) \xrightarrow{(t)^{\{l_1,l_2\}}:\{l_2\in\{l_1,l_2\}\}}$$
$$\Big(\{t/x\}^{\{l_1,l_2\}} \mid \lfloor P_1\rfloor_{l_1}^{l_2} \mid \lfloor Q_1\{t/x\}\rfloor_{l_2}^{l_1} \mid \lfloor(z).R_1\rfloor_{l_3}\Big).$$

This example includes only one broadcast step. There is no replication. First the rule (Ext BroadSend2) is applied, then, the rule (Ext BroadRecv1) is applied, which models $Q$ receives $t$, because $l_2 \in \{l_1, l_2\}$.

### 4.7.2 Example for multiple broadcast send and receive

The next example is a bit more complicated that includes replication and multiple broadcasts. The network can be seen in the Figure 5. Here, both nodes $P$ and $Q$ are the neighbors of $R$. First $P$ broadcasts $t_1$ then $Q$ broadcasts $t_2$. Node $R$ is under replication which intuitively means that it repeatedly listens for messages.

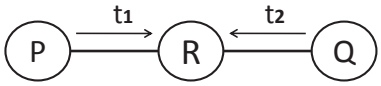


Figure 5: Another example network

$$\Big(\lfloor\langle t_1\rangle.P\rfloor_{l_1}^{\{l_3\}} \mid \lfloor\langle t_2\rangle.Q\rfloor_{l_2}^{\{l_3\}} \mid \lfloor!(y).R\rfloor_{l_3}^{\{l_1,l_2\}}\Big) \xrightarrow{\nu x.\langle x\rangle:\overline{l_1}\{l_3\}}$$
$$\Big(\{t_1/x\}^{\{l_1,l_3\}} \mid \lfloor P\rfloor_{l_1}^{\{l_3\}} \mid \lfloor\langle t_2\rangle.Q\rfloor_{l_2}^{\{l_3\}} \mid \lfloor!(y).R\rfloor_{l_3}^{\{l_1,l_2\}}\Big) \xrightarrow{(t_1)^{\{l_1,l_3\}}:\{l_3\in\{l_1,l_3\}\}}$$
$$\Big(\{t_1/x\}^{\{l_1,l_3\}} \mid \lfloor P\rfloor_{l_1}^{\{l_3\}} \mid \lfloor\langle t_2\rangle.Q\rfloor_{l_2}^{\{l_3\}} \mid \lfloor R\{t_1/x\} \mid !(y).R\rfloor_{l_3}^{\{l_1,l_2\}}\Big) \xrightarrow{\nu z.\langle z\rangle:\overline{l_2}\{l_3\}}$$
$$\Big(\{t_1/x\}^{\{l_1,l_3\}} \mid \{t_2/z\}^{\{l_2,l_3\}} \mid \lfloor P\rfloor_{l_1}^{\{l_3\}} \mid \lfloor Q\rfloor_{l_2}^{\{l_3\}} \mid \lfloor R\{t_1/x\} \mid !(y).R\rfloor_{l_3}^{\{l_1,l_2\}}\Big) \xrightarrow{(t_2)^{\{l_2,l_3\}}:\{l_3\in\{l_2,l_3\}\}}$$
$$\Big(\{t_1/x\}^{\{l_1,l_3\}} \mid \{t_2/z\}^{\{l_2,l_3\}} \mid \lfloor P\rfloor_{l_1}^{\{l_3\}} \mid \lfloor Q\rfloor_{l_2}^{\{l_3\}} \mid \lfloor R\{t_2/z\} \mid R\{t_1/x\} \mid !(y).R\rfloor_{l_3}^{\{l_1,l_2\}}\Big).$$

Each labeled transition step is similar as in the previous example with the only difference that each rule is applied twice due to the two broadcast communications.

### 4.7.3 Example for mobility

The next example illustrate the mobility issue and message loss when the message sent by a node $N_1$ is not received by an another node $N_2$ because $N_2$ moved out of the transmission range of $N_1$. The scenario can be seen in the Figure 6. In this scenario, at first two nodes $N_1$ and $N_2$ have no connection after that the node $N_2$ gets into the transmission range of the node $N_1$. After this the message $t$ broadcasted by $N_1$ is received by $N_2$. In the next reduction step $N_2$ moves out of the transmission range of $N_1$, and then the message $t$ sent by $N_1$ is not intercepted by $N_2$, thus, $t$ is lost.

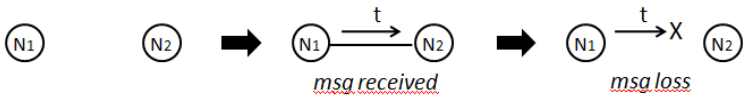


Figure 6: Example for mobility and message loss.

We let $N_1$ and $N_2$ be $\lfloor !\langle t\rangle.P_1\rfloor_{l_1}$, and $\lfloor !(x).P_2\rfloor_{l_2}$, respectively. At first, they are not connected. After the reduction relation $\rightarrow_{\{l_1 \bullet l_2\}}$ the node $N_2$ moves into the transmission range of the node $N_1$. Then node $N_1$ broadcasts $t$, which is then received by $N_2$. These steps are modelled by the labeled transitions $\stackrel{\nu x.\langle x\rangle:\overline{l_1}\{l_2\}}{\longrightarrow}$ and $\stackrel{(t)^{\{l_1,l_2\}}:\{l_2\in\{l_1,l_2\}\}}{\longrightarrow}$. After these labeled transition steps the operation of nodes $N_1$ and $N_2$ get into the states $P_1$ and $P_2\{t/x\}$, respectively. As the next step, node $N_2$ moved out of the transmission range of $N_1$. This modelled by the reduction relation $\rightarrow_{\{l_1 \circ l_2\}}$. At this time, node $N_1$ against broadcasts $t$, which now is not intercepted by $N_2$.

$$\left(\lfloor !\langle t\rangle.P_1\rfloor_{l_1} \mid \lfloor !(x).P_2\rfloor_{l_2}\right) \rightarrow_{\{l_1 \bullet l_2\}} \left(\lfloor !\langle t\rangle.P_1\rfloor_{l_1}^{l_2} \mid \lfloor !(x).P_2\rfloor_{l_2}\right) \stackrel{\nu x.\langle x\rangle:\overline{l_1}\{l_2\}}{\longrightarrow}$$

$$\left(\{t/x\}^{\{l_1,l_2\}} \mid \lfloor P_1 \mid !\langle t\rangle.P_1\rfloor_{l_1}^{l_2} \mid \lfloor !(x).P_2\rfloor_{l_2}\right) \stackrel{(t)^{\{l_1,l_2\}}:\{l_2\in\{l_1,l_2\}\}}{\longrightarrow}$$

$$\left(\{t/x\}^{\{l_1,l_2\}} \mid \lfloor P_1 \mid !\langle t\rangle.P_1\rfloor_{l_1}^{l_2} \mid \lfloor P_2\{t/x\} \mid !(x).P_2\rfloor_{l_2}\right) \rightarrow_{\{l_1 \circ l_2\}}$$

$$\left(\{t/x\}^{\{l_1,l_2\}} \mid \lfloor P_1 \mid !\langle t\rangle.P_1\rfloor_{l_1} \mid \lfloor P_2\{t/x\} \mid !(x).P_2\rfloor_{l_2}\right) \stackrel{\nu y.\langle y\rangle:\overline{l_1}\{l_2\}}{\longrightarrow}$$

$$\left(\{t/y\}^{\{l_1\}} \mid \{t/x\}^{\{l_1,l_2\}} \mid \lfloor P_1 \mid !\langle t\rangle.P_1\rfloor_{l_1} \mid \lfloor P_2\{t/x\} \mid !(x).P_2\rfloor_{l_2}\right).$$

In the next subsection we introduce the static equivalent and labeled bisimilarity in the new context, in particular, on Mobile Ad-hoc Networks. We can use them to prove security properties of secure on-demand source routing protocols, as well as reasoning about attacks.

## 4.8 Attacker knowledge base, static equivalence, labeled bisimilarity

We let $\mathcal{L}(N)$ be the set of identifier $l_i$'s in the network $N$, again we recall that each $l_i$ is a unique name in the network, and identify each node of the network. Then we let $connect_j(\mathcal{L}(N)) \in \mathcal{C}(N)$ be a set of all links in the j-th topology of the network $N$. $\mathcal{C}(N)$ is the set of all possible topologies of $N$.

Recall that an extended network is composed of active substitution with range and plain networks as follow:

$$E{=}\nu\tilde{n}.\,(\{^{t_1}/_{x_1}\}^{\sigma_1} \,|\, \{^{t_2}/_{x_2}\}^{\sigma_2} \,|\, \ldots \,|\, \{^{t_m}/_{x_m}\}^{\sigma_m} \,|N_1| \ldots |N_r)$$

The output of the extended network $E$ is defined by a *frame* $\varphi$, which is composed of name restrictions and a parallel composition of **all** active substitutions: $\varphi = \nu\tilde{n}.\,(\{^{t_1}/_{x_1}\} \,|\, \{^{t_2}/_{x_2}\} \,|\, \ldots \,|\, \{^{t_m}/_{x_m}\})$. We note that in $\varphi$ the ranges $\sigma_1, \ldots,$ $\sigma_n$ are removed from active substitutions with range.

Intuitively, the frame represents the output of the network. We note that while the attacker node knows only the messages sent by its neighbors the wireless environment knows all of the sent messages. By hearing everything the wireless environment can distinguish the operation of two networks. When an attack (against a routing protocol) is executed successfuly on a specific topology the wireless environment will be aware of it, because it can distinguish the correct from the incorrect operations.

Taking into account that in our adversary model the attacker is weaker than the Dolev-Yao attacker in the sense that he cannot eavesdrop all the messages sent in the network but only messages from its neighbors. On the other hand, the attacker can perform numerous computation steps based on its knowledge. Adapting the notion of frame, the **accumulated knowledge base of the attacker** is defined as the frame with the identifier $l_a$ as parameter: $\varphi(l_a)$. The frame $\varphi(l_a)$ can be seen as the "subset" of the frame $\varphi$, because it contains only such active substitution(s) $\{^{t_i}/_{x_i}\}^{\sigma_i}$ $(i{\in}\{1,\ldots,m\})$ where $l_a \in \sigma_i$. That is,

$$\varphi(l_a){=}\nu\tilde{n}.\,\left(\{^{t_i}/_{x_i}\}^{\sigma_i} \,|\, \left\{^{t_j}/_{x_j}\right\}^{\sigma_j} \,|\, \ldots \,|\, \{^{t_k}/_{x_k}\}^{\sigma_k}\right),$$

where $l_a \in \sigma_i$, $l_a \in \sigma_j$,..., $l_a \in \sigma_k$, and $\{i, j, \ldots, k\} \subseteq \{1, 2 \ldots, n\}$.

**Definition 7.** *Two terms $t_1$ and $t_2$ are equal in a frame $\varphi$, and write $[t_1 = t_2]\,\varphi$, if and only if $\varphi \equiv \nu\tilde{n}.\omega$, $t_1\omega = t_2\omega$, and $\{\tilde{n}\} \cap (fn(t_1) \cup fn(t_2)) = \emptyset$ for some names $\tilde{n}$ and substitution $\omega$.*

**Definition 8.** *Two closed frames $\varphi$ and $\psi$ are statically equivalent, and write $\varphi \approx_s \psi$, when $dom(\varphi) = dom(\psi)$ and when, for all terms $t_1$ and $t_2$, we have $[t_1 = t_2]\,\varphi$ if and only if $[t_1 = t_2]\,\psi$.*

*We say that two closed extended networks are statically equivalent, and write $E_1 \approx_s E_2$, when their frames are statically equivalent.*

**Lemma 1.** *Static equivalence is closed by structural equivalence, by reduction, and by application of closing evaluation contexts C[-].*

*Proof.* Due to the definition of frame we use in static equivalent is the same as in the applied $\pi$-calculus. The proof is also the same as in [11]. □

The advantage of the static equivalence is that it does not depend on the arbitrary environment of processes. Instead in order to check the validity of the equivalent it is enough to verify the frames we already know.

Unlike the spi-calculus [1], which is designed for reasoning about security protocols, where one has to define the relation $\Re$ that pairs two processes between which he wants to prove observational equivalence. Furthermore, one has to define the cipher environment and takes care about the attacker's (process $R$ that comes into contact with the two processes as a parallel composition) condition after each relation step, that is, one has to prove that secret key materials still not be obtained by the attacker (that is, names represents secret keys not become a free name of $R$).

Finally, we define the labeled bisimilarity in the context of Mobile Ad-Hoc Network. The advantage of the labeled bisimilarity is that it does not depend on an arbitrary context but only on the frames which is well-known after each transition step.

In order to make the definition be intuitive in the context of Mobile Ad-Hoc Networks, in the next definition without corrupting the correctness we assume that

$E_1$ consists of one plain network $N_1$, and $E_2$ consists of one plain network $N_2$. Again, we note that we are considering only the reasoning about the attacks discussed in Section 2, hence, we assume that the topology remains unchanged during attacks

**Definition 9.** *Labeled bisimilarity ($\approx_l^N$) is the largest symmetric relation $\Re$ on closed extended networks such that $E_1 \ \Re \ E_2$ implies: $\mathcal{L}(N_1) = \mathcal{L}(N_2)$ and $connect_i(\mathcal{L}(N_1)) = connect_j(\mathcal{L}(N_2))$, and*

1. *$E_1 \approx_s E_2$;*
2. *if $E_1 \longrightarrow E_1'$, then $E_2 \longrightarrow^* E_2'$ and $E_1' \ \Re \ E_2'$ for some $E_2'$; (This is the induction based on internal reductions);*
3. *if $E_1 \xrightarrow{\alpha} E_1'$ and $fv(\alpha) \subseteq dom(E_1)$ and $bn(\alpha) \cap fn(E_2) = \emptyset$; then $E_2 \longrightarrow^* \xrightarrow{\alpha} \longrightarrow^* E_2'$ and $E_1' \ \Re \ E_2'$ for some $E_2'$. (This is the induction based on labeled relations). Here $\alpha$ can be a broadcast, an unicast, or a receive action.*

Intuitively, this means that the outputs of the two networks of same topology cannot be distinguished during their operation. 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 (inputs) something then the node $l$ in $E_2$ outputs (inputs) 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, or more formally, a transitive and reflexive closure of $\rightarrow$.

**Definition 10.** *Given $E_1$ and $E_2$ such that $\mathcal{L}(N_1) = \mathcal{L}(N_2)$. We say that $E_1$ and $E_2$ are labeled bisimilar if they are labeled bisimilar in every same topology. That is, $\forall connect_i \in \mathcal{C}(N_1)$, $\forall connect_j \in \mathcal{C}(N_2)$, such that $connect_i(\mathcal{L}(N_1)) = connect_j(\mathcal{L}(N_2))$ : $E_1 \approx_l^N E_2$.*

## 4.9 Example on modelling the attacker knowledge base

Let us consider the example topologies in the Figure 7. Next we demonstrate how to model that the attacker node collects information, namely, how the attacker builds its knowledge base during the route discovery phase.

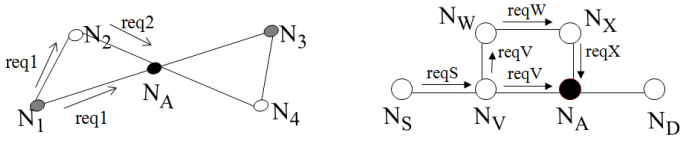


Figure 7: On the left is the topology in which an attack is found against the SRP protocol: Node $N_1$ initiates the route discovery towards node $N_3$, and node $N_A$ is the attacker node. On the right is the topology in which an attack is found against the Ariadne protocol: Node $N_S$ initiates the route discovery towards node $N_D$, and node $N_A$ is the attacker node.

First, we consider the scenario on the left. Let $N_1$ be $\lfloor P_1 \rfloor_{l_1}^{\{l_2,l_a\}}$, $N_2$ be $\lfloor P_2 \rfloor_{l_2}^{\{l_1,l_a\}}$, $N_3$ be $\lfloor P_3 \rfloor_{l_3}^{\{l_a,l_4\}}$, $N_4$ be $\lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}}$, and $N_A$ be $\lfloor P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}}$. Then the topology on the left of the Figure 7 is specified as:

$$netw_1 \stackrel{def}{=} \lfloor P_1 \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor P_2 \rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}}$$

After the broadcasting of the request $req_1$ by node $N_1$ $netw_1$ gets into the state $netw_1'$.

$$netw_1 \stackrel{\nu x.\langle x \rangle : \overline{l_1}\{l_2,l_a\}}{\longrightarrow} netw_1', \text{ where}$$

$$netw_1' \stackrel{def}{=} \{^{req_1}/_x\}^{\{l_2,l_a\}} \mid \lfloor P_1' \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor P_2 \rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}}$$

The active substitution with range $\{^{req_1}/_x\}^{\{l_2,l_a\}}$ means that the attacker node intercepts the request $req_1$ because $l_a \in \{l_2, l_a\}$. Thus, at this time the knowledge base of the attacker node is increased with $req_1$. The process $P_1'$ is the process we reach from $P_1$ after broadcasting $req_1$.

Then, after $N_2$ broadcasts $req_2$ the network $netw_1'$ reaches the state $netw_1''$:

$$netw_1' \xrightarrow{\nu y.\langle y\rangle:\overline{l_2}\{l_1,l_a\}} netw_1'', \text{ where}$$

$$netw_1'' \stackrel{def}{=} \{^{req_2}/_y\}^{\{l_1,l_a\}} \mid \{^{req_1}/_x\}^{\{l_2,l_a\}} \mid \lfloor P_1' \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor P_2' \rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}}$$

The active substitution with range $\{^{req_2}/_y\}^{\{l_1,l_a\}}$ means that the attacker node intercepts the request $req_2$ because $l_a \in \{l_1, l_a\}$. Hence, at this point the knowledge of the attacker node has been extended with $req_1$ and $req_2$. Formally, let us assume that the initial knowledge of the attacker is zero, then we have $\varphi(l_a) = \{^{req_2}/_y\} \mid \{^{req_1}/_x\}$.

The scenario on the right side in the Figure 7 can be described in the same manner as in the scenario on the left side.

*We note that this ability of the sr-calculus is novel compared to CMAN and the $\omega$-calculus. With this ability the sr-calculus can be used to directly modelling the attacks found against the SRP and Ariadne protocols, which is not the case in CMAN and the $\omega$-calculus.*

## 4.10 Example on labeled bisimilarity ($\approx_l^N$)

Let us consider the example network and scenario $netw_1$ in the Figure 8 that is similar to the case in the Figure 5. In addition, let us consider the another network $netw_2$, which has the same topology (node identifiers and neighborhood) as $netw_1$. However, in $netw_2$ the first node has the internal operation $\langle t_3\rangle.P'$ in which the message $t_3$ is broadcast instead of $t_1$ as in the case of $\langle t_1\rangle.P$, where $t_1 \neq t_3$.

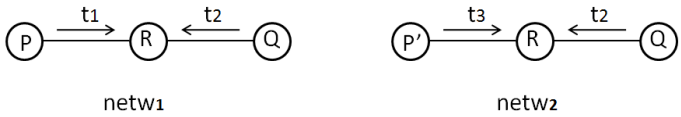


Figure 8: The two networks $netw_1$ and $netw_2$ have the same topology but the operations of the leftmost nodes differ (P and P').

$$netw_1 \stackrel{def}{=} \left(\lfloor\langle t_1\rangle.P\rfloor_{l_1}^{\{l_3\}} \mid \lfloor\langle t_2\rangle.Q\rfloor_{l_2}^{\{l_3\}} \mid \lfloor !(y).R\rfloor_{l_3}^{\{l_1,l_2\}}\right) \xrightarrow{\nu x.\langle x\rangle:\overline{l_1}\{l_3\}}$$
$$\left(\{t_1/x\}^{\{l_3\}} \mid \lfloor P\rfloor_{l_1}^{\{l_3\}} \mid \lfloor\langle t_2\rangle.Q\rfloor_{l_2}^{\{l_3\}} \mid \lfloor !(y).R\rfloor_{l_3}^{\{l_1,l_2\}}\right) \xrightarrow{(t_1)^{\{l_3\}}:\{l_2\in\{l_3\}\}}$$
$$\left(\{t_1/x\}^{\{l_3\}} \mid \lfloor P\rfloor_{l_1}^{\{l_3\}} \mid \lfloor\langle t_2\rangle.Q\rfloor_{l_2}^{\{l_3\}} \mid \lfloor R\{t_1/x\} \mid !(y).R\rfloor_{l_3}^{\{l_1,l_2\}}\right) \xrightarrow{\nu z.\langle z\rangle:\overline{l_2}\{l_3\}}$$
$$\left(\{t_1/x\}^{\{l_3\}} \mid \{t_2/z\}^{\{l_3\}} \mid \lfloor P\rfloor_{l_1}^{\{l_3\}} \mid \lfloor Q\rfloor_{l_2}^{\{l_3\}} \mid \lfloor R\{t_1/x\} \mid !(y).R\rfloor_{l_3}^{\{l_1,l_2\}}\right) \xrightarrow{(t_2)^{\{l_3\}}:\{l_2\in\{l_3\}\}}$$
$$\left(\{t_1/x\}^{\{l_3\}} \mid \{t_2/z\}^{\{l_3\}} \mid \lfloor P\rfloor_{l_1}^{\{l_3\}} \mid \lfloor Q\rfloor_{l_2}^{\{l_3\}} \mid \lfloor R\{t_2/z\} \mid R\{t_1/x\} \mid !(y).R\rfloor_{l_3}^{\{l_1,l_2\}}\right).$$

$$netw_2 \stackrel{def}{=} \left( \lfloor \langle t_3 \rangle.P' \rfloor_{l_1}^{\{l_3\}} \mid \lfloor \langle t_2 \rangle.Q \rfloor_{l_2}^{\{l_3\}} \mid \lfloor !(y).R \rfloor_{l_3}^{\{l_1,l_2\}} \right) \stackrel{\nu x.\langle x \rangle : \overline{l_1}\{l_3\}}{\longrightarrow}$$
$$\left( \{t_3/x\}^{\{l_3\}} \mid \lfloor P' \rfloor_{l_1}^{\{l_3\}} \mid \lfloor \langle t_2 \rangle.Q \rfloor_{l_2}^{\{l_3\}} \mid \lfloor !(y).R \rfloor_{l_3}^{\{l_1,l_2\}} \right) \stackrel{(t_3)^{\{l_3\}} : \{l_2 \in \{l_3\}\}}{\longrightarrow}$$
$$\left( \{t_3/x\}^{\{l_3\}} \mid \lfloor P' \rfloor_{l_1}^{\{l_3\}} \mid \lfloor \langle t_2 \rangle.Q \rfloor_{l_2}^{\{l_3\}} \mid \lfloor R\{t_1/x\} \mid !(y).R \rfloor_{l_3}^{\{l_1,l_2\}} \right) \stackrel{\nu z.\langle z \rangle : \overline{l_2}\{l_3\}}{\longrightarrow}$$
$$\left( \{t_3/x\}^{\{l_3\}} \mid \{t_2/z\}^{\{l_3\}} \mid \lfloor P' \rfloor_{l_1}^{\{l_3\}} \mid \lfloor Q \rfloor_{l_2}^{\{l_3\}} \mid \lfloor R\{t_1/x\} \mid !(y).R \rfloor_{l_3}^{\{l_1,l_2\}} \right) \stackrel{(t_2)^{\{l_3\}} : \{l_2 \in \{l_3\}\}}{\longrightarrow}$$
$$\left( \{t_3/x\}^{\{l_3\}} \mid \{t_2/z\}^{\{l_3\}} \mid \lfloor P' \rfloor_{l_1}^{\{l_3\}} \mid \lfloor Q \rfloor_{l_2}^{\{l_3\}} \mid \lfloor R\{t_2/z\} \mid R\{t_1/x\} \mid !(y).R \rfloor_{l_3}^{\{l_1,l_2\}} \right).$$

It is easy to see that $netw_1$ and $netw_2$ are **not** labeled bisimilar, because their outputs, that is their frames $\{t_1/x\}|\{t_2/z\}$ and $\{t_3/x\}|\{t_2/z\}$ can be distinguished because $t_1 \neq t_3$.

# 5 Attacker's ability and knowledge

The computation ability of the attacker is an unchanged set, denoted by $\mathcal{A}_c$, of constructor functions such as computing encryption, hash, and digital signature, compose a message tuple etc. The knowledge of the attacker is composed of initial knowledge (denoted by $\mathcal{K}_{init}$) and gained knowledge (denoted by $\mathcal{K}_{gain}$). Typically, $\mathcal{K}_{init}$ often contains the node IDs of the neighborhood of the attacker, and pre-shared keys. Hence, formally, when modeling the initial knowledge we initiate the frame $\varphi(l_a)$ with the substitution (with the range $\{l_a\}$, hence, not available for the honest nodes) of the initial knowledge on new variables. Thereafter, the frame $\varphi(l_a)$ is periodically extended with new knowledge (i.e., $\mathcal{K}_{gain}$), and whenever the attacker compute a message for his purpose it can use its whole knowledge and computation ability.

The computation ability of the attacker node is the set $\mathcal{B}$ of functions such as *encrypt(t,k)*, *hash(t)*, *mac(t,k)*, *sign(t,k)*, etc. To capture the attacker's ability for message verification, set $\mathcal{B}$ also contains equations from $\sum$, such as *dec(enc(x, y),y) = x* and *checksign(sign(x, y),pk(y)) = true* for a special constant *true*. In the processes of the attacker, the parameters of these functions and equations can only those that appear in $\varphi(l_a)$.

To make the behavior of the attacker systematic, we assume that the attacker tries all the possible moves (selecting possible functions, equations with the available parameters). To reduce the number of possibilities we can explicitly add the type-respect binding of the parameters to functions and equations. For instance, in *sign(t,k)*, from $\varphi(l_a)$ only the terms of type DATA can be bound to $t$, and only terms of type PRIVATEKEY can be bound to $k$. Moreover, in case of source routing protocols, usually, the patterns (skeletons) of the accepted reply and request are known. Hence, by reasoning in a backward manner that which kind of message parts the attacker need to have in order to compose the reply or request that includes an invalid route but fullfils the pattern of accepted message, we can systematically make the analysis and greatly reducing the number of possibilities.

# 6 Application of the calculus

In this section we demonstrate the usability of the *sr*-calculus by modelling the SRP protocol and the attack scenario we discussed in Section 2.

In order to model secure on-demand source routing protocols for mobile ad-hoc networks we introduce the following required constructor functions and destructor applications.

We start with the discussion of the construction function tuple and the next and previous functions related to tuple, then we discuss the MAC function.
***tuple***: The constructor function **tuple** models a tuple of $n$ terms $t_1$, $t_2$,..., $t_n$. We write the function as

$$tuple(t_1, t_2, \ldots, t_n)$$

We abbreviate it simply as $(t_1, t_2, \ldots, t_n)$ in the rest of the paper.

We introduce the destructor functions **i** that returns the i-th element of a tuple of n elements, where $i \in \{1, \ldots, n\}$:

$$i(t_1, t_2, \ldots, t_n) = t_i$$

***list***: The constructor function **list** models a list of $n$ terms $t_1$, $t_2$,..., $t_n$. We write the function as

$$list(t_1, t_2, \ldots, t_n)$$

We abbreviate it as $[t_1, t_2, \ldots, t_n]$ in the rest of the paper. We note that a list can be empty, that is $n = 0$, and we denote the empty list as [ ].

Then the destructor applications **next** and **prev** are introduced for modelling the next and the previous element of a particular element in the list. Each function has two arguments, a first is a *list* and the second is the term of which we want to know its next and previous element in the given list. If there is no such element in the list or it has no next or previous element then it returns a constant symbol ***undefined***. The sort system may enforce that *next* and *prev* are applied only to list.

$$\begin{array}{l} next([t_1, \ldots, t_i, t_{i+1}, \ldots t_n], t_i) = t_{i+1}, \\ next([t_1, \ldots, t_n], t_n) = undefined \\ next([t_1, \ldots, t_n], t_i) = undefined, \text{ if } t_i \text{ does not occur in the list,} \\ prev([t_1, \ldots, t_{i-1}, t_i, \ldots t_n], t_i) = t_{i-1}, \\ prev([t_1, \ldots, t_n], t_1) = undefined \\ prev([t_1, \ldots, t_n], t_i) = undefined, \text{ if } t_i \text{ does not occur in the list.} \end{array}$$

We also introduce the functions **toendlist** and **toheadlist** that model the list with $n + 1$ elements by appending an element $t$ to the end (to the head) of a list $t_{list}$ of $n$ elements with same sort as $t$, respectively.

$$\begin{array}{l} toendlist\,([t_1, \ldots, t_n], t) = [t_1, \ldots, t_n, t] \\ toheadlist\,(t, [t_1, \ldots, t_n]) = [t, t_1, \ldots, t_n]. \end{array}$$

In the rest of the paper for convenient presentation we write $[t_1, \ldots, t_n, t]$ instead of $\mathit{toendlist}\,([t_1, \ldots, t_n], t)$. With the function $\mathit{toendlist}$ we can model lists as follows: $[l_1, l_2, \ldots, l_n] = \mathit{toendlist}(\ldots \mathit{toendlist}(\mathit{toendlist}([\,], l_1), l_2) \ldots l_n)$.

Functions *first*, and *last* represents the first, and the last element of *List*, respectively.

$$\mathit{first}([t_1, \ldots, t_n]) = t_1$$
$$\mathit{last}([t_1, \ldots, t_n]) = t_n.$$

Finally we model the keyed hash or MAC function with symmetric key $k$ with the binary function **mac**. The

$$\mathit{mac}(t_1, t_2).$$

function that computes the message authentication code of message $t_1$ using secret key $t_2$. The shared key between node $l_i$ and $l_j$ is modelled by function $k(l_i, l_j)$.

## 6.1 Modelling the SRP protocol and the attack

The scenario in Section 2 is modelled by the extended network defined as:

$$\mathit{netw} \stackrel{def}{=} (\lfloor P_1 \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor P_2 \rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor !P_A \rfloor_{l_a}^{\{l_1,l_2l_3,l_4\}} \mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}}).$$

where $N_1{=}\lfloor P_1 \rfloor_{l_1}^{\{l_2,l_a\}}$, $N_2{=}\lfloor P_2 \rfloor_{l_2}^{\{l_1,l_a\}}$, $A{=}\lfloor !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}}$, $N_3{=}\lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}}$, $N_4{=}\lfloor P_4 \rfloor_{l_4}^{l_al_3}$. Processes $P_1$,$P_2$,$!P_3$,$P_4$ model the operation of honest nodes while process $A$ model the operation of the attacker node as follows:

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

Intuitively, the node $l_1$ generates the route request message that includes the ID of source and target nodes, and the message authentication code $MAC_{13}$ computed using the shared key, broadcasts it and waits for the reply. When it 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 using its shared key. If all are correct then it signals term $\mathit{ACCEPT}$. The process $P_2$ models the operation of the node $N_2$ and is specified as follow:

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

$$
\begin{aligned}
WaitRep_2 \stackrel{def}{=}\ & (y_{rep})\,.[1(y_{rep}) = l_2][2(y_{rep}) = rep] \\
& [next\,(5(y_{rep}), l_2) \in \{l_1 l_a\}] \\
& in\ \langle (l_1, 2(y_{rep}), 3(y_{rep}), 4(y_{rep}), 5(y_{rep}), 6(y_{rep}))\rangle.
\end{aligned}
$$

Intuitively, on receiving a message it checks if it is a request, then appends its ID $l_2$ to the end of the list, re-broadcasts it and waits for a reply. When it receives the reply message it checks if the message is intended to it, it is a reply, the next ID in the list corresponds to neighbors and forwards the message to the destination node $l_1$. The process $P_4$ models the operation of the node $N_4$ and is specified as follow:

$$
\begin{aligned}
P_4 \stackrel{def}{=}\ & (z_{req})\,.[1(z_{req}) = req]. \\
& \langle (1(z_{req}), 2(z_{req}), 3(z_{req}), 4(z_{req}), [5(z_{req}), l_4])\rangle \\
& !\,WaitRep_4. \\
WaitRep_4 \stackrel{def}{=}\ & (z_{rep})\,.[1(z_{rep}) = l_2][2(z_{rep}) = rep] \\
& [prev\,(5(z_{rep}, l_4)) \in \{l_a l_3\}] \\
& let\ t_{idList} = 5(z_{rep})\ in\ let\ l_{prev} = prev\,(t_{idList}, l_4) \\
& in\ \langle (l_{prev}, 2(z_{rep}), 3(z_{rep}), 4(z_{rep}), 5(z_{rep}), 6(z_{rep}))\rangle.
\end{aligned}
$$

Intuitively, on receiving a message node $N_4$ checks if it is a request, then appends its identifier $l_4$ to the end of the list, re-broadcasts it and waits for a reply. When it receives the reply message it checks if the message is intended to it, it is a reply, the previous and next ID in the list corresponds to neighbors and forwards the message to the previous node $l_{prev}$ in the list. The process $P_4$ models the operation of the node $N_4$ and is specified as follow:

Finally, the operation of the destination node $N_3$, the process $P_3$, is modelled as:

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

Intuitively, on receiving the a message it checks if the message is a request, and it is the destination, and verifies the MAC embedded in the request using its shared key with $l_1$. If so then it creates a reply message and forwards it to the last node in the list.

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

The attacker overhears only messages sent by its neighbors. Let frame $\varphi(l_a)$ be $\{^{t_i}/_{x_i}\} \mid \{^{t_j}/_{x_j}\} \mid \ldots \mid \{^{t_k}/_{x_k}\}$. This represents the attacker's knowledge he

accumulates during the route discovery phase by eavesdropping. He combines this accumulated knowledge and initial knowledge to construct an attack. 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 $(x_1,\ldots,x_n)$ of variables, $\nu n$ means the attacker creates new data $n$. The function $f\,(\tilde{x}, T_{l^P}, n)$ represents the message the attacker generates from the eavesdropped messages that it receives by binding them to $\tilde{x}$, its initial knowledge and the newly generated data $n$, respectively. At first, $\tilde{x}$ is a single variable $x_a$.

As the next step, we define an ideal model of *netw*, written 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 a special function *consistent*$(List)$ that informs it about the correctness of the returned route. We define this ideal source node as follow:

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

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

**Definition 11.** *A routing protocol is said to be secure if for all extended networks $E$ and its corresponding ideal network $E_{spec}$, which includes an arbitrary attacker node, we have: $E \approx_l^N E_{spec}$.*

**Theorem 1.** *The SRP protocol is insecure.*

*Proof.* We will show that $netw \approx_l^N netw_{spec}$ does **not** hold besides the attacker $\mathcal{M}_A$ because the third point of the Definition 9 is violated. In order to do this we will show that there exist a sequence of labeled transitions and internal reduction relations that can be performed in case of *netw* but can not be performed in case of $netw_{spec}$. Formally, this means that the frames of *netw* and $netw_{spec}$ can be distinguished.

Let us see the following sequence of labeled transitions and reduction relations that *netw* can perform: First the source node $l_1$ broadcast the route request message $(req, l_1, l_3, MAC_{13}, [\,])$ to initiates the discovery of the route towards the node $l_3$.

$$\left(\lfloor P_1 \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor P_2 \rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}}\right) \xrightarrow{\nu x.\langle x\rangle:\overline{l_1}\{l_2,l_a\}}$$

$(\{^{(req,l_1,l_3,MAC_{13},[\,])}/_x\}^{\{l_1,l_2,l_a\}} \mid \lfloor P_1' \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor P_2 \rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}}$
$\mid \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}})$ ($\rightarrow \times 2$ (EXT BroadRecv1))

The active substitution with range $\{^{(req,l_1,l_3,MAC_{13},[\,])}/_x\}^{\{l_1,l_2,l_a\}}$ means that after $l_1$ broadcasts the message $(req, l_1, l_3, MAC_{13}, [\,])$ it is available for itself and its neighbors, the nodes $l_1$ and $l_a$. The process $P_1'$ is $!\,WaitRep_1$. After applying two times the rule (EXT BroadRecv1) *netw* reaches the following state:

$(\{^{(req,l_1,l_3,MAC_{13},[\,])}/_x\}^{\{l_1,l_2,l_a\}} \mid \lfloor !\,WaitRep_1 \rfloor_{l_1}^{\{l_2,l_a\}}$
$\mid \lfloor P_2 \{^{(req,l_1,l_3,MAC_{13},[\,])}/_x\} \rfloor_{l_2}^{\{l_1,l_a\}}$
$\mid \lfloor P_A \{^{(req,l_1,l_3,MAC_{13},[\,])}/_x\} \mid !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}}$
$\mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}})$ ($\rightarrow \times 2$ (Red P-Let))

Intuitively, this means that node $l_2$ and the attacker node $l_a$ receive the broadcasted message. After broadcasting the message $(req, l_1, l_3, MAC_{13}, [\,])$ node $l_1$ reaches to the state $!\,WaitRep_1$ and after receiving the route request message the nodes $l_2$ and $l_a$ are going to broadcast their message. The node $l_2$ is going to broadcast the request message $(req, l_1, l_3, MAC_{13}, [l_2])$ and the attacker node $(req, l_1, l_3, MAC_{13}, [l_2, n, l_4])$. ($\rightarrow \times 2$ (Red P-Let)) means the application of the reduction relation rule (Red P-Let) twice.

$(\{^{(req,l_1,l_3,MAC_{13},[\,])}/_x\}^{\{l_1,l_2,l_a\}} \mid \lfloor !\,WaitRep_1 \rfloor_{l_1}^{\{l_2,l_a\}}$
$\mid \lfloor \langle 1(x), 2(x), 3(x), 4(x), [5(x), l_2] \rangle .!\,WaitRep_2 \rfloor_{l_2}^{\{l_1,l_a\}}$
$\mid \lfloor \langle 1(x), 2(x), 3(x), 4(x), [l_2, n, l_4] \rangle \mid !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}}$
$\mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}})$ $\xrightarrow{\nu y.\langle y \rangle : \overline{l_a}\{l_1,l_2,l_3,l_4\}}$

As we mentioned earlier the broadcast sends are choosed non-deterministically they are going to executed at the same time. We assume that in this labeled transition trace the attacker node outputs its message before node $l_2$. After broadcasting $(req, l_1, l_3, MAC_{13}, [l_2, n, l_4])$ *netw* reaches the following state:

$(\{^{(req,l_1,l_3,MAC_{13},[\,])}/_x\}^{\{l_1,l_2,l_a\}} \mid \{^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])}/_y\}^{\{l_a,l_1,l_2,l_3,l_4\}}$
$\mid \lfloor !\,WaitRep_1 \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor \langle 1(x), 2(x), 3(x), 4(x), [5(x), l_2] \rangle .!\,WaitRep_2 \rfloor_{l_2}^{\{l_1,l_a\}}$
$\mid \lfloor 0 \mid !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}}) \equiv_{Struct\ P-Par1}$
($\rightarrow \times 4$ (EXT BroadRecv1))

We note that at this time the frame of *netw* is $(\{^{(req,l_1,l_3,MAC_{13},[\,])}/_x\}^{\{l_1,l_2,l_a\}}$
$\mid \{^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])}/_y\}^{\{l_a,l_1,l_2,l_3,l_4\}}$, which represents the output messages so far. After applying the rules (Struct P-Par1) and 4 times the rule (EXT BroadRecv1) *netw* reaches the following state:

$(\{^{(req,l_1,l_3,MAC_{13},[\,])}/_x\}^{\{l_1,l_2,l_a\}} \mid \{^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])}/_y\}^{\{l_a,l_1,l_2,l_3,l_4\}}$

$$
\begin{array}{l}
\quad \mid \lfloor WaitRep_1 \left\{ {}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])} /_y \right\} \mid !WaitRep_1 \rfloor_{l_1}^{\{l_2,l_a\}} \\
\quad \mid \lfloor P_2' \left\{ {}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])} /_y \right\} \rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \\
\quad \mid \lfloor P_3 \left\{ {}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])} /_y \right\} \rfloor_{l_3}^{\{l_a,l_4\}} \\
\quad \mid \lfloor P_4 \left\{ {}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])} /_y \right\} \rfloor_{l_4}^{\{l_a,l_3\}}) \rightarrow^*
\end{array}
$$

Here the process $P_2'$ is $\langle 1(x), 2(x), 3(x), 4(x), [5(x), l_2] \rangle .!WaitRep_2$. After receiving the message $(req, l_1, l_3, MAC_{13}, [l_2, n, l_4])$ broadcasted by the attacker node, nodes $l_1$ and $l_2$ drops it since the verification they make on it fails. This is modelled by the nil process.

$$
\begin{array}{l}
(\left\{ {}^{(req,l_1,l_3,MAC_{13},[\ ])} /_x \right\}^{\{l_1,l_2,l_a\}} \mid \left\{ {}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])} /_y \right\}^{\{l_a,l_1,l_2,l_3,l_4\}} \\
\quad \mid \lfloor 0 \mid !WaitRep_1 \rfloor_{l_1}^{\{l_2,l_a\}} \\
\quad \mid \lfloor P_2' \rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor \langle l_4, rep, 2(y), 3(y), 5(y), MAC_{31} \rangle \mid !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \\
\quad \mid \lfloor \langle 1(y), 2(y), 3(y), 4(y), [5(y), l_4] \rangle .!WaitRep_4 \rfloor_{l_4}^{\{l_a,l_3\}}) \\
\quad \longrightarrow^*, \equiv_{Struct\ P-Par1} \times 2, \xrightarrow{\nu x'.\langle x' \rangle : \overline{l_2}\{l_1,l_a\}}
\end{array}
$$

After applying a sequence of reduction relations that models the verification made on the message and applying the rule (Struct P-Par1) twice the nil processes are eliminated, and applying the labeled transition $\xrightarrow{\nu x'.\langle x' \rangle : \overline{l_2}\{l_1 l_a\}}$, we have that *netw* reaches the following state:

$$
\begin{array}{l}
(\left\{ {}^{(req,l_1,l_3,MAC_{13},[\ ])} /_x \right\}^{\{l_1,l_2,l_a\}} \mid \left\{ {}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])} /_y \right\}^{\{l_a,l_1,l_2,l_3,l_4\}} \\
\quad \mid \left\{ {}^{(req,l_1,l_3,MAC_{13},[l_2])} /_{x'} \right\}^{\{l_2,l_1,l_a\}} \mid \lfloor !WaitRep_1 \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor !WaitRep_2 \rfloor_{l_2}^{\{l_1,l_a\}} \\
\quad \mid \lfloor !P_A \rfloor_{l_a}^{l_1 l_2 l_3 l_4} \mid \lfloor \langle l_4, rep, 2(y), 3(y), 5(y), MAC_{31} \rangle \mid !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \\
\quad \mid \lfloor \langle 1(y), 2(y), 3(y), 4(y), [5(y), l_4] \rangle .!WaitRep_4 \rfloor_{l_4}^{\{l_a,l_3\}}) \\
\quad \xrightarrow{\nu z.\langle z \rangle : \overline{l_3}\{l_a,l_4\}}
\end{array}
$$

After applying a sequence of reduction relations that models the verification made on the message and applying the rule (Struct P-Par1) twice the nil processes are eliminated, and applying the labeled transition $\xrightarrow{\nu z.\langle z \rangle : \overline{l_3}\{l_a l_4\}}$, which models that the destination node $l_3$ accepts the message sent by the attacker node and sends back a reply message. Again, we assume that in this transition trace node $l_3$ sends its message before node $l_4$:

$$
\begin{array}{l}
(\left\{ {}^{(req,l_1,l_3,MAC_{13},[\ ])} /_x \right\}^{\{l_1,l_2,l_a\}} \mid \left\{ {}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])} /_y \right\}^{\{l_a,l_1,l_2,l_3,l_4\}} \\
\quad \mid \left\{ {}^{(req,l_1,l_3,MAC_{13},[l_2])} /_{x'} \right\}^{\{l_2,l_1,l_a\}} \mid \left\{ {}^{(l_4,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})} /_z \right\}^{\{l_3,l_a,l_4\}} \\
\quad \mid \lfloor !WaitRep_1 \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor P_2' \rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor 0 \mid !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \\
\quad \mid \lfloor \langle 1(y), 2(y), 3(y), 4(y), [5(y), l_4] \rangle .!WaitRep_4 \rfloor_{l_4}^{\{l_a,l_3\}})
\end{array}
$$

$\equiv_{StructP-Par1}$, ($\rightarrow \times 2$ (EXT BroadRecv1))

The frame of *netw* at this time is

$\left\{{}^{(req,l_1,l_3,MAC_{13},[\,])}/_{x}\right\}^{\{l_1,l_2,l_a\}} \mid \left\{{}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])}/_{y}\right\}^{\{l_a,l_1,l_2,l_3,l_4\}}$
$\mid \left\{{}^{(req,l_1,l_3,MAC_{13},[l_2])}/_{x'}\right\}^{\{l_2,l_1,l_a\}} \mid \left\{{}^{(l_4,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}/_{z}\right\}^{\{l_3,l_a,l_4\}}$. After applying the rules (Struct P-Par1) and (EXT BroadRecv1) twice we have:

$(\left\{{}^{(req,l_1,l_3,MAC_{13},[\,])}/_{x}\right\}^{\{l_1,l_2,l_a\}} \mid \left\{{}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])}/_{y}\right\}^{\{l_a,l_1,l_2,l_3,l_4\}}$
$\quad \mid \left\{{}^{(req,l_1,l_3,MAC_{13},[l_2])}/_{x'}\right\}^{\{l_2,l_1,l_a\}} \mid \left\{{}^{(l_4,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}/_{z}\right\}^{\{l_3,l_a,l_4\}}$
$\quad \mid \lfloor !\,WaitRep_1 \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor !\,WaitRep_2 \rfloor_{l_2}^{\{l_1,l_a\}}$
$\quad \mid \left\lfloor P_A \left\{{}^{(l_4,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}/_{z}\right\} \right.$ $\mid !P_A \rfloor_{l_a}^{l_1 l_2 l_3 l_4}$
$\quad \mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \left\lfloor P_4' \left\{{}^{(l_4,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}/_{z}\right\} \right\rfloor_{l_4}^{\{l_a,l_3\}}) \equiv$

Here $P_4'$ is $\langle 1(y), 2(y), 3(y), 4(y), [5(y), l_4]\rangle.!\,WaitRep_4$. At this point, after receiving the reply message $(l_4, rep, l_1, l_3, [l_2, n, l_4], MAC_{31})$ node $l_4$ drops it because $l_4$ still has not output the request corresponding to this reply. However, the attacker node intercepts the reply.

$(\left\{{}^{(req,l_1,l_3,MAC_{13},[\,])}/_{x}\right\}^{\{l_1,l_2,l_a\}} \mid \left\{{}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])}/_{y}\right\}^{\{l_a,l_1,l_2,l_3,l_4\}}$
$\quad \mid \left\{{}^{(req,l_1,l_3,MAC_{13},[l_2])}/_{x'}\right\}^{\{l_2,l_1,l_a\}} \mid \left\{{}^{(l_4,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}/_{z}\right\}^{\{l_3,l_a,l_4\}}$
$\quad \mid \lfloor !\,WaitRep_1 \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor !\,WaitRep_2 \rfloor_{l_2}^{\{l_1,l_a\}}$
$\quad \mid \lfloor \langle l_1, 2(z), 3(z), 4(z), 5(z), 6(z)\rangle \mid !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}}$
$\quad \mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4' \rfloor_{l_4}^{\{l_a,l_3\}}) \xrightarrow{\nu w.\langle w\rangle : \overline{l_a}\{l_1 l_2 l_3 l_4\}}$

Then the attacker node $l_a$ forwards the reply message $(l_1, rep, l_1, l_3, [l_2, n, l_4], MAC_{31})$ to node $l_1$ in the name of node $l_2$. This message can be overheared by the neighbors of the node $l_a$.

$(\left\{{}^{(req,l_1,l_3,MAC_{13},[\,])}/_{x}\right\}^{\{l_1,l_2,l_a\}} \mid \left\{{}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])}/_{y}\right\}^{\{l_a,l_1,l_2,l_3,l_4\}}$
$\quad \mid \left\{{}^{(req,l_1,l_3,MAC_{13},[l_2])}/_{x'}\right\}^{\{l_2,l_1,l_a\}} \mid \left\{{}^{(l_4,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}/_{z}\right\}^{\{l_3,l_a,l_4\}}$
$\quad \mid \left\{{}^{(l_1,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}/_{w}\right\}^{\{l_a,l_1,l_2,l_3,l_4\}}$
$\quad \mid \lfloor !\,WaitRep_1 \rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor !\,WaitRep_2 \rfloor_{l_2}^{\{l_1,l_a\}}$
$\quad \mid \lfloor 0 \mid !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P_4' \rfloor_{l_4}^{\{l_a,l_3\}}) \equiv_{PAR\text{-}0}, \rightarrow^*$

The sent reply message $(l_1, rep, l_1, l_3, [l_2, n, l_4], MAC_{31})$ is then overheared by nodes $l_1$, $l_2$, $l_3$, and $l_4$. However, nodes $l_2$, $l_3$, and $l_4$ drops it because they are not the addressee but node $l_1$.

$(\left\{{}^{(req,l_1,l_3,MAC_{13},[\,])}/_{x}\right\}^{\{l_1,l_2,l_a\}} \mid \left\{{}^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])}/_{y}\right\}^{\{l_a,l_1,l_2,l_3,l_4\}}$
$\quad \mid \left\{{}^{(req,l_1,l_3,MAC_{13},[l_2])}/_{x'}\right\}^{\{l_2,l_1,l_a\}} \mid \left\{{}^{(l_4,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}/_{z}\right\}^{\{l_3,l_a,l_4\}}$

$$
\begin{aligned}
&| \left\{^{(l_1,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}\,/_w\right\}^{\{l_a,l_1,l_2,l_3,l_4\}} \\
&| \lfloor WaitRep_1 \left\{^{(l_1,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}\,/_w\right\} \mid !WaitRep_1\rfloor_{l_1}^{\{l_2,l_a\}} \\
&| \lfloor !WaitRep_2\rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor !P_A\rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor !P_3\rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P'_4\rfloor_{l_4}^{\{l_a,l_3\}}) \rightarrow^*
\end{aligned}
$$

After receiving the message $(l_1, rep, l_1, l_3, [l_2, n, l_4], MAC_{31})$ the source node $l_1$ makes verification steps on it. According to the operation of the protocol all verification steps made by $l_1$ pass and thus the term $ACCEPT$ is being to output.

$$
\begin{aligned}
&(\left\{^{(req,l_1,l_3,MAC_{13},[\,])}\,/_x\right\}^{\{l_1,l_2,l_a\}} \mid \left\{^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])}\,/_y\right\}^{\{l_a,l_1,l_2,l_3,l_4\}} \\
&\quad \mid \left\{^{(req,l_1,l_3,MAC_{13},[l_2])}\,/_{x'}\right\}^{\{l_2,l_1,l_a\}} \mid \left\{^{(l_4,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}\,/_z\right\}^{\{l_3,l_a,l_4\}} \\
&\quad \mid \left\{^{(l_1,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}\,/_w\right\}^{\{l_a,l_1,l_2,l_3,l_4\}} \\
&\quad \mid \lfloor \langle ACCEPT \rangle \mid !WaitRep_1\rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor !WaitRep_2\rfloor_{l_2}^{\{l_1,l_a\}} \\
&\quad \mid \lfloor !P_A\rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor !P_3\rfloor_{l_3}^{\{l_a,l_4\}} \mid \lfloor P'_4\rfloor_{l_4}^{\{l_a,l_3\}}) \xrightarrow{\nu v.\langle v\rangle:\overline{l_1}\{l_2 l_a\}}
\end{aligned}
$$

After the source node $l_1$ receives the rely message sent by the attacker it sees that it is the addressee. Hence, it makes verification steps. All verification steps pass so that it outputs the special term $ACCEPT$.

$$
\begin{aligned}
&(\left\{^{(req,l_1,l_3,MAC_{13},[\,])}\,/_x\right\}^{\{l_1,l_2,l_a\}} \mid \left\{^{(req,l_1,l_3,MAC_{13},[l_2,n,l_4])}\,/_y\right\}^{\{l_a,l_1,l_2,l_3,l_4\}} \\
&\quad \mid \left\{^{(req,l_1,l_3,MAC_{13},[l_2])}\,/_{x'}\right\}^{\{l_2,l_1,l_a\}} \mid \left\{^{(l_4,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}\,/_z\right\}^{\{l_3,l_a,l_4\}} \\
&\quad \mid \left\{^{(l_1,rep,l_1,l_3,[l_2,n,l_4],MAC_{31})}\,/_w\right\}^{\{l_a,l_1,l_2,l_3,l_4\}} \mid \left\{^{ACCEPT}\,/_v\right\}^{l_1 l_2 l_a} \\
&\quad \mid \lfloor 0 \mid !WaitRep_1\rfloor_{l_1}^{\{l_2,l_a\}} \mid \lfloor !WaitRep_2\rfloor_{l_2}^{\{l_1,l_a\}} \mid \lfloor !P_A\rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} \mid \lfloor !P_3\rfloor_{l_3}^{\{l_a,l_4\}} \\
&\quad \mid \lfloor P'_4\rfloor_{l_4}^{\{l_a,l_3\}}).
\end{aligned}
$$

We note that in this section we intend to demonstrate the applicability of the *sr*-calculus for reasoning about secure routing protocols and for shorter presentation purpose the proof illustrates the following attack scenario: When the attacker node receives the request message ($req$, $l_1$, $l_3$, $MAC_{13}$,[ ]) from node $l_1$, it creates some new fake node indentifier $n$, then adds $l_2$, $n$ and the identifier of $N_4$, $l_4$ to the list [ ], 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 makes by $l_3$. Then, node $l_3$ generates the reply ($l_4$, $rep$, $l_1$, $l_3$,$MAC_{31}$) and sends 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 $ACCEPT$ by the $\xrightarrow{\overline{l_1}\{l_2 l_a\}}$ transition relation. However, in $netw_{spec}$ node $l_1$ does not accept the returned route, thus, $ACCEPT$ is not output. Formally, at this point $netw_{spec}$ cannot perform the transition $\xrightarrow{\nu v.\langle v\rangle:\overline{l_1}\{l_2 l_a\}}$, which violates the third point of Definition 9. In this proof $netw_{spec}$, which is the ideal version of $netw$, can perform the same labeled transitions and reduction relations as $netw$ except the last transition $\xrightarrow{\nu v.\langle v\rangle:\overline{l_1}\{l_2 l_a\}}$.

Finally, we note that we can easily extend the proof so that it illustrates the scenario in which the attacker 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 adds $n$ and the identifier of $N_4$, $l_4$ to the list [ ], and re-broadcasts ($req$, $l_1$,$l_3$,$MAC_{13}$,$[l_2, n, l_4]$). The rest part of the attack scenario is the same as the scenario above. □

# 7 Weaker definition of security: up to barb $\Downarrow$ $ACCEPT$

One can feel that the notion of labeled bisimilarity may be too strict for defining security because it requires the existence of same outputs in the two networks. In fact, we only need that for the same topology and attacker behavior trace the real and specification networks always do the same with respect to the barb $ACCEPT$. Namely, if the specification can/cannot emit the term $ACCEPT$ then so do the real system, and vice versa. Then this must be true for all the possible topologies and attackers.

We note that this kind of definition is not necessarily weaker than the labeled bisimilarity. It depends on the definition of the specification version, and how much it differs from the real system.

# 8 A systematic proof technique based on backward deduction

The proof shown in Section 6.1 is based on a forward search/reasoning, namely, we specify a certain network topology and "simulate" the operation of the protocol on this topology. The main drawback of this proof technique is that we need to take into account a huge number of possible behavior scenarios as well as many possible network topologies.

We develope a more systematic proof technique, that enables us to reason about the security of routing protocols in a more efficient way. This proof technique is based on backward reasoning, namely, we start with the assumption that the source has accepted an invalid route, and based on the definition of the protocol we reason backward step-by-step to find out how could this happen. In case we get a contradiction it means that the starting assumption could not be valid, and the protocol is secure. Like in the forward search technique, this proof technique is also based on Definition 9 but with backward reasoning.

For this backward reasoning method, we define an ideal and a real system a bit differently. First of all, the source node in both systems will output the function term $accept(t_{list})$, instead of outputting the constant term $ACCEPT$. This means that the source has accepted the returned list $t_{list}$.

The procedure of the backward reasoning (backward deduction) is performed by following the protocol, we step back node-by-node from the destination to

the source through some route, which can either be the route represented by $t_{list}$ in $accept(t_{list})$ or an another route. Formally, let us denote the state when the source has accepted the list $t_{list}$ by the network $E_{accept(t_{list})}$, and the state when the source initiates a request by $E_{reqinit}$. The backward deduction is based on performing labelled transitions backward from $E_{accept(t_{list})}$ to $E_{reqinit}$. We use the upper index *real* ($E^{real}_{reqinit}$, $E^{real}_{accept(t_{list})}$) and *ideal* ($E^{ideal}_{reqinit}$, $E^{ideal}_{accept(t_{list})}$) to denote the corresponding network states in the real and ideal systems, respectively.

$$E_{accept(t_{list})} \ {}^{*}\!\longleftarrow \overset{\alpha_1}{\longleftarrow}{}^{*}\!\longleftarrow \ldots {}^{*}\!\longleftarrow \overset{\alpha_n}{\longleftarrow}{}^{*}\!\longleftarrow E_{reqinit},$$

where $\alpha_1,\ldots,\ \alpha_n$ can be a broadcast, an unicast, or a receive action. For instance, the following backward deduction trace

$$\begin{array}{l} E_{dstsentREP} \ {}^{\nu z.\langle z\rangle:\ \overline{l_{dst}}\{\ l_{int},\ldots\}}\!\!\longleftarrow {}^{*}\!\longleftarrow E_{dstrecvdREQ} \ {}^{(t_{req})^{\sigma 2}:\{l_{dst}\in\sigma 2\}}\!\!\longleftarrow E_{intsentREQ} \\ \qquad {}^{\nu y.\langle y\rangle:\ \overline{l_{int}}\{\ l_{dst},l_{src},\ \ldots\}}\!\!\longleftarrow {}^{*}\!\longleftarrow E_{intrecvdREQ} \ {}^{(t_{reqinit})^{\sigma 1}:\{l_{int}\in\sigma 1\}}\!\!\longleftarrow E_{srcsentREQINIT} \\ \qquad\qquad {}^{\nu x.\langle x\rangle:\ \overline{l_{src}}\{\ l_{int},\ldots\}}\!\!\longleftarrow {}^{*}\!\longleftarrow E_{srcreqinit}, \end{array}$$

where the frame of $E_{dstsentREP}$ contains the substitution $\{t_{rep}/z\}$, and the frames of $E_{intsentREQ}$ and $E_{intsentREQINIT}$ contain $\{t_{req}/y\}$ and $\{t_{reqinit}/x\}$, respectively. $\sigma 1$ and $\sigma 2$ are the neighborhood of $l_{int}$ and $l_{dst}$, respectively. Intuitively, the trace

$$E_{dstsentREP} \ {}^{\nu z.\langle z\rangle:\ \overline{l_{dst}}\{\ l_{int},\ldots\}}\!\!\longleftarrow {}^{*}\!\longleftarrow E_{dstrecvdREQ} \ {}^{(t_{req})^{\sigma 2}:\{l_{dst}\in\sigma 2\}}\!\!\longleftarrow E_{intsentREQ}$$

says that in order to $l_{dst}$ can send the reply $t_{rep}$ it should have received the request $t_{req}$ from $l_{int}$. The trace

$$E_{intsentREQ} \ {}^{\nu y.\langle y\rangle:\ \overline{l_{int}}\{\ l_{dst},l_{src},\ \ldots\}}\!\!\longleftarrow {}^{*}\!\longleftarrow E_{intrecvdREQ} \ {}^{(t_{reqinit})^{\sigma 1}:\{l_{int}\in\sigma 1\}}\!\!\longleftarrow E_{srcsentREQINIT}$$

says that in order to node $l_{int}$ can send the request $t_{req}$ it should have received the request $t_{reqinit}$ from $l_{src}$. Finally, the trace

$$E^{src}_{sentREQINIT} \ {}^{\nu x.\langle x\rangle:\ \overline{l_{src}}\{\ l_{int},\ldots\}}\!\!\longleftarrow {}^{*}\!\longleftarrow E^{src}_{reqinit}$$

says that in order to node $l_{src}$ can send the request $t_{reqinit}$ it should have been able to composed this request.

This trace corresponds to the following scenario: On the route $l_{src} - l_{int} - l_{dst}$,

1. $l_{src}$ broadcasts $t_{reqinit}$;
2. $l_{int}$ received $t_{reqinit}$, performs calculations, and broadcasts $t_{req}$;
3. $l_{dst}$ received $t_{req}$, performs verifications, and returns $t_{rep}$.

In particular, the backward deduction is as follows:

1. At the beginning, we assume that the source node has accepted the list $t_{list}$, which means that the function term $accept(t_{list})$ has been output by $l_{src}$. Formally, at first the frames of both $E^{real}_{accept(t_{list})}$ and $E^{ideal}_{accept(t_{list})}$ contain only the substitution

   $$\{accept(t_{list}) \ / \ x_{accept}\},$$

   During the remaining deduction steps, we will reason about how the presence of this substitution could happen. Based on the protocol specification and the message format of the request and reply, we analyze which messages should have been sent by which nodes that eventually leads to the acceptance of $t_{list}$. At some point during the backward reasoning procedure, if we found that a message $t_{msg}$ should have been sent, then the frame at that point will be extended with the substitution $\{t_{msg} \ / \ x_{msg}\}$.

2. The backward deduction *terminates* through a given route $[l_{src}, \ldots, l_{dst}]$ if by stepping back node-by-node in this list, we successfully get back to $E_{reqinit}$ (i.e., the initial request is output by the source $l_{src}$). More precisely, this means that the frame of $E_{reqinit}$ is extended with the substitution $\{t_{reqinit} \ / \ x_{reqinit}\}$, where $t_{reqinit}$ is the request sent by the source.

Basically, the backward deduction always terminates, because for every ID list $t_{list}$ in $accept(t_{list})$, we can get back to the state $E_{reqinit}$, by assuming that $t_{list}$ is an existing route and we step backward on this route. However, to detect an attack we focus on the case when $t_{list}$ is not a valid route, and we reason about how could this invalid route be accepted (if this is the case). Let us assume that $t_{list}$ in $accept(t_{list})$ represents an invalid route. In case the backward reasoning procedure terminates through the route other than $t_{list}$, it means that the routing protocol is insecure because the attacker can achieve that the invalid route $t_{list}$ is accepted, while if the deduction procedure can only terminate through the route $t_{list}$ then we get a contradiction since $t_{list}$ cannot be invalid, otherwise, we could not traverse back through it. Hence, in the latter case the protocol is secure.

In this backward deduction procedure, in order to perform a systematic proof based on Definition 9, we distinguish the ideal system and the real system in the following way: In the ideal system, the source always can check the correctness of the returned route $t_{list}$ by using the special function $consistent(t_{list})$, and only outputs $accept(t_{list})$ if $t_{list}$ is a correct route from the source to the destination. Hence, we define the ideal system such that the backward deduction can only terminate through the route $t_{list}$, and the deduction based on the other routes will be forbidden to perform the last transition representing the broadcast of the initial request:

$$E_{srcsentREQINIT} \xleftarrow{\nu x.\langle x\rangle:\ \overline{l_{src}}\sigma} * \longleftarrow E_{srcreqinit} \qquad (last\text{-}TRANS)$$

Intuitively, this means that in the ideal system every accepted route must be a valid route. However, this is not the case in the real system, where the attacker(s) can achieve that an invalid route will be accepted. We model this by allowing the possibility for the deduction to terminate (i.e., allowing the last transition) either through the route $t_{list}$ or any other routes.

To prove the security of on-demand source routing protocols based on backward deduction we apply the Definition 9 in reverse direction, specifically:

**Lemma 2.** *Let $E^{real}_{accept(t_{list})}$ and $E^{ideal}_{accept(t_{list})}$ be the real and the ideal specification variants of a (on-demand source) routing protocol Prot in the sr-calculus. The protocol Prot is said to be secure if for all the possible routes represented by the list $t_{list}$, the following holds:*

1. $E^{real}_{accept(t_{list})} \approx_s E^{ideal}_{accept(t_{list})}$;
2. *if* $E^{real}_{accept(t_{list})} \longleftarrow E^{realprv}_{accept(t_{list})}$, *then* $E^{ideal}_{accept(t_{list})} \stackrel{*}{\longleftarrow} E^{idealprv}_{accept(t_{list})}$ *and* $E^{realprv}_{accept(t_{list})} \Re E^{idealprv}_{accept(t_{list})}$ *for some* $E^{idealprv}_{accept(t_{list})}$;
3. *if* $E^{real}_{accept(t_{list})} \stackrel{\alpha}{\longleftarrow} E^{realprv}_{accept(t_{list})}$ *and* $fv(\alpha) \subseteq dom(E^{realprv}_{accept(t_{list})})$ *and* $bn(\alpha) \cap fn(E^{ideal}_{accept(t_{list})}) = \emptyset$; *then* $E^{ideal}_{accept(t_{list})} \stackrel{*}{\longleftarrow} \stackrel{\alpha}{\longleftarrow} \stackrel{*}{\longleftarrow} E^{idealprv}_{accept(t_{list})}$ *and* $E^{realprv}_{accept(t_{list})} \Re E^{idealprv}_{accept(t_{list})}$ *for some* $E^{idealprv}_{accept(t_{list})}$, *where $\alpha$ can be a broadcast, an unicast, or a receive action.*

Intuitively, Lemma 2 says that starting from the states $E^{real}_{accept(t_{list})}$ and $E^{ideal}_{accept(t_{list})}$, if the two backward deduction procedures cannot be distinguished from each other then the routing protocol is secure. In other words, the deduction of the ideal system can simulate every deduction trace of the real system. The rationality behind this Lemma is based on the fact that the backward deduction of the ideal and the real systems differ only at the last transition (*last-TRANS*), which is allowed in the ideal case only when the deduction trace (so far) conforms with the list $t_{list}$, while it is allowed in the real case for every possible trace. Hence, the deduction of the ideal system, $E^{ideal}_{accept(t_{list})}$, can simulate the deduction of $E^{real}_{accept(t_{list})}$ if in both cases (*last-TRANS*) can only be performed when the deduction trace confoms with $t_{list}$. This means that $t_{list}$, which is accepted at the end of the route discovery, must be valid. The ideal system is defined such that the accepted list $t_{list}$ must always be valid, and if we deduce that this is also true for the real system, then the routing protocol is secure.

## 8.1 General specification of on-demand source routing protocols

In this subsection, a general and simplified specification of the on-demand source routing protocols is given, which is well-suited for the backward deduction technique. The specification is based on the *sr*-calculus, but instead of defining specific network topologies, we provide a general specificaion that includes the

specification of a source, a destination, and some intermediate nodes, regardless of the topology.

On-demand source routing protocols have an important flavour that each node usually has the same uniform internal operation: during route discovery each node can play a role of a source node, or an intermediate node, or a destination node. Leveraging this beneficial characteristic, we need to specify only the operation of three nodes instead of all of the nodes in the network, which is more comfortable.

$$E_{routing} \stackrel{def}{=} \lfloor !P_{src} \rfloor_{l_{src}}^{\sigma_{src}} \mid \prod_{i \in 1,\ldots,n} \left\lfloor !P_{int}^{i} \right\rfloor_{l_{int}^{i}}^{\sigma_{int}^{i}} \mid \lfloor !P_{dst} \rfloor_{l_{dst}}^{\sigma_{dst}}.$$

where $N_{src} = \lfloor !P_{src} \rfloor_{l_{src}}^{\sigma_{src}}$, $N_{int}^{i}$, $N_{int}^{i} = \left\lfloor !P_{int}^{i} \right\rfloor_{l_{int}^{i}}^{\sigma_{int}^{i}}$, represents the i-th intermediate node, and $\prod_{i \in 1,\ldots,n} \left\lfloor !P_{int}^{i} \right\rfloor_{l_{int}^{i}}^{\sigma_{int}^{i}}$ represents the parallel composition of $n$ intermediate nodes $\left\lfloor !P_{int}^{i} \right\rfloor_{l_{int}^{i}}^{\sigma_{int}^{i}}$, for $i \in \{1, \ldots, n\}$, $N_{dst}$=$\lfloor !P_{dst} \rfloor_{l_{dst}}^{\sigma_{dst}}$. Processes $P_{src}$, $P_{int}$, $P_{dst}$ model the operation of honest nodes. We do not need to include explicitly the behavior of the attacker node(s). The attackers is modelled by the wireless environment in an implicit way, which can be seen as a cooperation of several attackers. In case an attack scenario is detected, the specific place of the attacker(s) is determined based on the messages it (they) intercepts or sends during the scenario. The number of the intermediate nodes, $n$, is also determined based on the specific detected attack scenario.

We note that the specific structure of each process depends on the specific routing protocol.

## 8.2 The backward deduction algorithm

At the beginning, a reply including an invalid route $t_{list} = [l_1, \ldots, l_n]$ is assumed to be accepted by the source. Afterwards, we follow the way of this reply in a backward manner. The possible paths of this reply is investigated by reasoning about the nodes and edges through which this reply and the corresponding request should have traversed during the route discovery. On searching for the possible paths of the reply and request backward, whenever the attacker node is reached, it means that the reply or request has been forwarded (and may be modified) by the attacker node. If this is the case, at this point we are aware of the information of what message should the attacker forward to be accepted later, that is, what messages should the attacker generate in order to perform a successful attack. This is then followed by examining how the attacker can generate these messages.

The attacker is able to compose a reply or request message using its computational ability and knowledge base. We note that while the computation ability of the attacker is fix, its knowledge base is continually updated during the route discovery. Hence, by backward reasoning we mean the reasoning about three issues: (i) Can the attacker generate each part of the message based only on its computational ability and initial knowledge? (ii) Which messages should

the attack node intercept in case it cannot set up a whole reply/request based solely on its computational ability and initial knowledge? (iii) How the topology should be formed such that the attacker is able to intercept the required message parts?

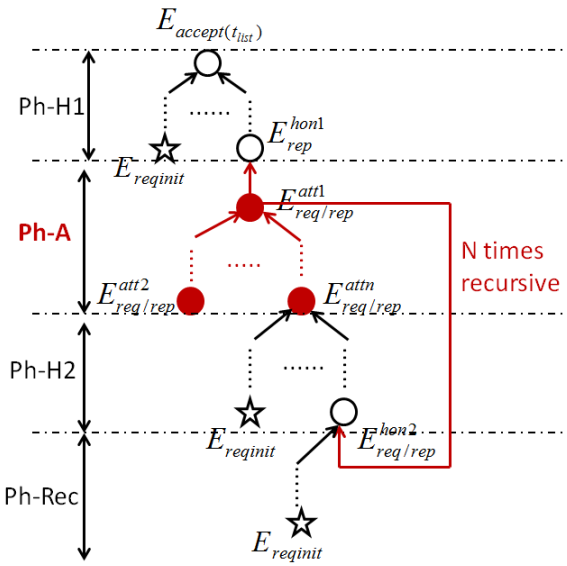


Figure 9: The figure illustrates the main phases of the backward deduction procedure. The circles and asterisks represent the states of the deduction. Each state $state_i$ is a network $E^i_{req/rep}$, which we get from $E_{accept(t_{list})}$ after performing a series of transitions. The asterisks represent the terminal states, which is $E_{reqinit}$. Each phase may contain one or several states. After state $E^{hon2}_{req/rep}$ we can get into the phases Ph-A and Ph-H2 again, because there can be several attacker nodes or the attacker can take place in an interleaving route discovery session.

In phase *Ph-H1* we investigate how the request and reply propagate. The rightmost branch involve the interference of the attacker, and examines how the reply traverses from the attacker to the source. The backward deduction may not involve the attacker, and considers the scenario when the reply and request messages are sent only by honest nodes. This latter case is illustrated by the leftmost branch. The asterisk represents the last state $E_{reqinit}$, where the deduction terminates. Not that during a backward reasoning only one branch is chosen, until we reach $E_{reqinit}$.

The reasoning about how the attacker could generate an incorrect reply *Rep* or request *Req* which leads to a successful attack, takes place in phase *Ph-A*. In *Ph-A*. we examine how the attacker could generate the Req/Rep message that leads to a successful attack. In particular, how the attacker can obtain or compute all the parts of the request or reply messages. The attacker(s) can

obtain each part of a request/reply by either computes it based on the available information and the computation ability, or receives/intercepts a message that contains this part. In the latter case, the deduction procedure is continued with the phase *Ph-H2*, where we check how the attackers could receives/intercepts those messages. Similar to *Ph-H1*, in *Ph-H2* the remaining derivation may or may not involve more attacker's interference in the rightmost and lefmost branches, respectively. The phase *Ph-Rec* represents a recursive application of the attacker phase *Ph-A*. Typically, if $N = 2$ then there is one attacker and we are considering its interference behavior in both the reply and request parts. The case of $N > 2$ is for analysing the possibility of several attackers and interleaving attacks.

In the "honest" phases *Ph-H1*, *Ph-H2* the request and reply messages are forwarded by only the honest nodes. These phases include labelled transitions that models the broadcast send and receive actions by honest nodes, as well as the silent transitions for the (not the visible) verification steps made on the received message. Every step during the backward deduction in *Ph-H1* and *Ph-H2* is based on the specification of the honest nodes $N_{src}$, $N^i_{int}$ and $N_{dst}$.

**The attacker phase *Ph-A***: In the attacker phase *Ph-A*, if we found that the attacker must have sent the reply $t_{attrep}$ or the request $t_{attreq}$ to be successful, then in the rest steps we deduce how this message could have been composed. This phase include labelled transitions that applies when the attacker receives/intercepts or sends a message, and silent transitions that models the computations that are performed by the attacker.

An attack scenario is found when the deduction terminates (i.e., the state $E_{reqinit}$ is reached) through an route differ from $t_{list}$.

We let both $t_{attrep}$ and $t_{attreq}$ have the form $(head; v_1; \ldots; [List]; \ldots; v_k)$, which is true in most source routing protocols. The head part, *head*, is the tuple ($rreq/rrep : T_{req/rep}$, $s : T_{str}$, $d : T_{str}$, $sID : T_{str}$), where the first element has REQ/REP type, the remaining three elements have string type. The reason for giving the string type instead of node ID type or name type is that we want to give mode possibility to the attackers to replace these elements with some data of other types. The list $[List]$ is also given a string type, while $v_1$, $\ldots$, $v_n$ are additional protocol specific parts, which typically are some (crypto) functions computed on one portion of the message. For instance, in case of the SRP protocol we have one function part which is the MAC, while the Ariadne protocol includes hash and digital signature functions.

During the backward reasoning we attempt to find attacks with minimal number of transition steps.

1. First of all, we examines whether the attack could be performed if the attacker has forwarded the request/reply $t_{req/rep}$ unchanged. This means that in the Figure 9, the networks (or states) $E^{att1}_{req/rep}$ and $E^{attn}_{req/rep}$ are the same, and basically, the phase *Ph-A* contains only one state $E^{att}_{req/rep}$. If with this condition, the deduction terminates through an route differ from $t_{list}$, then an attack scenario is detected. In particular, when an attack scenario is detected we successfully prove the violation of Lemma 2.

2. Otherwise, if the deduction in the first point can terminate only through $t_{list}$, we return to examine how the attacker can obtain or compute each part of $t_{req/rep}$, where $t_{req/rep} = (head; v_1; \ldots; [List]; \ldots; v_k)$. An attack scenario is detected only when every elements of $t_{req/rep}$ can be computed/obtained by the attacker(s).

   We define *priority/weight* on terms of different types. We distinguish three classes of priority. The keyed crypto functions such as digital signatures, MAC function, public and symmetric key encryption have the highest priority. The next highest priority in line is assigned to keyless crypto functions such as one-way hash function. The lowest priority is given to non-crypto functions and data construct such as list and node IDs. Within the same priority terms are classified by *weight*, which specifies the number of variables and constants in them. The more subterms (names, variables, functions) are included the larger the weight is.

   Let $W$ be a set that contains the terms to be examined whether they can be computed or obtained by the attackers.

   (I.) First, $t_{req/rep}$ is decomposed and the resulted parts *head*, $v_1$, ..., $[List]$, $v_k$ are put into $W$.

   (II.) If there still are unexamined terms in $W$ we choose one of the highest priority group of terms, and within this group we start with the term that has highest weight that has not been examined before.

   (III.) For the name/constant terms $t_n$, we check whether they belong to the attacker's knowledge base, namely, $t_n \in \mathcal{K}_A$. For each function term $t_f$ we examine if can the attacker compute it using the current knowledge base $\mathcal{K}_A$ (e.g., keys for crypto functions). If there is a term $t_{n/f}$ in $W$ that the attacker cannot compute/obtain based on its current knowledge base then we go on with step (IV.).

   (IV.) For each term $t_{n/f}$ cannot compute by the attacker, we analyse how the attacker can intercept/receive this term from either a honest node or an another attacker node. In particular, we replace (in a type conform way) some part of $(head; v_1; \ldots; [List]; \ldots; v_k)$ with $t_{n/f}$, then we reason about how this modified message could propagate during the route discovery. If for every $t_{n/f}$ the deduction can terminate only through $t_{list}$, then the protocol is secure, otherwise, an attack scenario is found.

During the deduction procedure, whenever we get into a loop, namely, we reach the term that has already been examined before, then we finish that branch of deduction to avoid infinite loop. During the backward deduction, we also keep track of the topology $\mathcal{T}_{top}$, which is the topology belongs to an attack scenario (if any). At first, $\mathcal{T}_{top}$ does not contain any edges, and first state $E_{accept(t_{list})}$ is

$$E_{accept(t_{list})} \stackrel{def}{=} \{accept(t_{list})/x_{accept}\}^{\sigma_{src} \cup \{l_a\}} \mid \lfloor !P_{src} \rfloor_{l_{src}}^{\sigma_{src}} \mid \prod_{i \in 1,\ldots,n} \lfloor !P_{int}^i \rfloor_{l_{int}^i}^{\sigma_{int}^i} \mid \lfloor !P_{dst} \rfloor_{l_{dst}}^{\sigma_{dst}}.$$

where $\sigma_{src}$, $\sigma^i_{int}$ and $\sigma_{dst}$ are empty. During the backward deduction, if at one point we found that to make the source accepts $t_{list}$, the source $l_{src}$ should send a message $t_{req/rep}$ to $l_i$, then we update $\mathcal{T}_{top}$ with the new edge between $l_{src}$ and $l_i$. Further, we add $l_i$ to the neighborhood of $l_{src}$, $\sigma_{src}$, and add $l_{src}$ to $\sigma_i$ (assuming bi-directional edges). $\{accept(t_{list})/x_{accept}\}^{\sigma_{src}\cup\{l_a\}}$ says that the output $accept(t_{list})$ is available to the neghborhood of $l_{src}$ and the attacker. We add $l_a$ to each substitution, because by default we assume that the attacker nodes can be everywhere in the network. The exact number and location of the attackers depends on the deduction path. Similarly, the terminal state $E_{reqinit}$ is specified as follows:

$$E_{reqinit} \stackrel{def}{=} \{accept(t_{list})/x_{accept}\}^{\sigma_{src}\cup\{l_a\}}, \ldots, \{t_{reqinit}/x_{reqinit}\}^{\sigma_{src}\cup\{l_a\}} \mid \lfloor !P_{src} \mid P^{reqinit}_{src} \rfloor^{\sigma_{src}}_{l_{src}} \mid \prod_{i\in 1,\ldots,n} \lfloor !P^i_{int} \rfloor^{\sigma^i_{int}}_{l^i_{int}} \mid \lfloor !P_{dst} \rfloor^{\sigma_{dst}}_{l_{dst}}.$$

In $E_{reqinit}$, the frame is extended with the substitution $\{t_{reqinit}/x_{reqinit}\}^{\sigma_{src}\cup\{l_a\}}$, and $P^{reqinit}_{src}$ is the process we get after $t_{reqinit}$ has been broadcast in $P_{src}$.

# 9 Automating the verification

In this section, we present a novel automatic technique based on deductive model-checking specifically for verifying secure routing protocols for wireless ad-hoc networks.

There are numerous former works address the problem of verifying routing protocols. Some of them use existing model-checking tools such as SPIN [14] and Uppaal [6] to either verifying loop properties of routing protocols [24, 22, 23]. Other works apply the tools SPIN [14] and Uppaal [6] to verify the security property of secure routing protocols [5]. However, using these tools to model and verify secure routing protocols is very circumstancial due to they are not directly designed specifically for this purpose. They cannot be used to directly model cryptographic primitives and cryptographic operations, broadcast communication, neighborhood, and attacker in wireless environment. For instance, in [5, 18] broadcast communication is given as a series of uni-sends, or cryptographic primitives are modelled as a series of bits.

In contrast to these works, our work focuses primarily on modelling and verifying secure routing protocols for wireless ad-hoc networks in the presence of compromised nodes. In particular, we propose an automatic verification method based on logic and the deduction is based on resolution. In addition, our method can model broadcast communication, neighborhood and cryptographic primitives.

Our technique was inspired by the method used in the Proverif automatic verification tool proposed for verifying security protocols [7], however, as opposed to [7] it is designed for verifying *routing* protocols and includes numerous novelties such as the modelling of broadcast communications, neighborhood, transmission range, and considers an attacker model specific to wireless ad hoc networks instead of the Dolev-Yao attacker model.

One important difference between the modelling of secure routing protocols and security protocols is that while in security protocols each communicating entity can has different internal operation and structure, in case of secure routing protocols each communication entity usually has the same uniform structure: During the route discovery each node can be (i) source node, or (ii) intermediate node, or (iii) destination node. Hence when using the Proverif tool to model secure routing protocols, in case of the network including $n$ nodes, the user has to describe the operation of all $n$ nodes despite the fact that they are the same up to renaming variables and names. In contrast, in our method the user is required to specify only the "general" operation of nodes, which represents any node.

## 9.1 The concept of the verification method

In the ProVerif verification tool [7] the input of the tool is the protocol description in the syntax of the applied $\pi$-calculus. The tool then translates this to Prolog rules to make automatic reasoning.

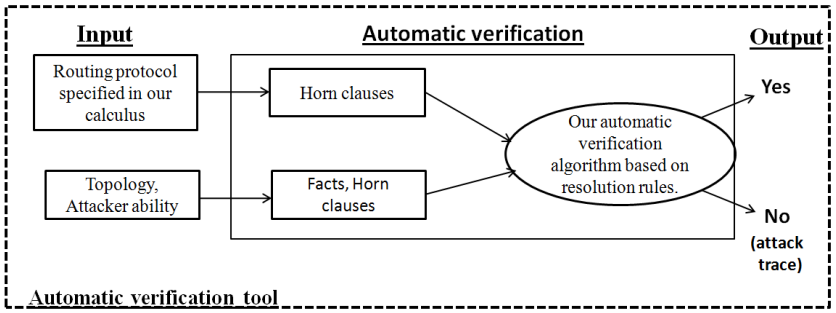


Figure 10: The concept of our automatic verification method.

Following this concept, in our verification method the operation of routing protocols are specified in the *syntax of processes* in the *sr*-calculus. This is then translated to Horn-clauses using translation rules. This set of clauses is called as *protocol rules*. In addition, the current topology and the initial knowledge of the attacker node are specified as a set of facts, while the computation ability of the attacker node is specified as the set of Horn-clauses. The clauses that specify the attacker computation ability are called as *attacker rules*. The deductive algorithm is based on the resolution steps over these clauses and facts.

## 9.2 From the calculus to Horn clauses

| | |
|---|---|
| $p ::=$ | *patterns* |
| $x^p, y^p, z^p$ | *variables* |
| $l_i^p$, $i \in \{1, \ldots, k\}$ | *honest node identifier* |
| $l_{att}^p$ | *attacker node identifier* |
| $req^p$, $rep^p$ | *special tags* |

| | |
|---|---|
| $i,\ i \in \{1,\ldots,m\}$ | *session identifier* |
| $s,\ s \in \{honest, advr\}$ | *special pattern that is used in the fact accept(s)* |
| $n\,[p_1,\ldots,p_k]$ | *data* |
| $f\,(p_1,\ldots,p_t)$ | *constructor application* |

Variables $x$, $y$, $z$ in the calculus are translated to patterns $x^p$, $y^p$, $z^p$, respectively. Node ID $l_n$ and $l_{att}$ are translated to $l_n^p$ and $l_{att}^p$, respectively. The patterns correspond to terms *req* and *rep* are $req^p$ and $rep^p$, respectively. We note that $l_n^p$, $l_{att}^p$, $req^p$, and $rep^p$ are atomic values (i.e., constants). Different session identifiers $i$ are associated to each session of processes under replication. Thanks to the session identifiers, we distinguish data created in different copies of processes, so different names in the process calculus are represented by different patterns. $s$ is a special variable that takes values from the set {*honest, advr*}, where *honest* and *advr* are constants. Finally, 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. *We note that patterns* $l_i^p$, $l_{att}^p$, $s$, $req^p$ *and* $rep^p$ *are novel compared with the ProVerif tool.*

The protocol and the computation ability of the attacker are modelled by the implication rules (clauses) of the form $F_1 \wedge \ldots \wedge F_n \rightarrow C$, where $F_i\ i \in \{1,\ldots,n\}$ and $C$ are facts that are defined as predicate application $pred(p_1,\ldots,p_m)$. The left side of the rule is called as hypothesis while the right side is called as conclusion.

Similarly as the model proposed in [10], our method includes the notion of wireless medium (wireless environment) that models the wireless environment. The wireless medium eavesdrops all communication in the network. The attacker node ables to catch only nodes sent by its neightbors. We model the wireless medium, and the attacker by the predicates **wm** and **att**, respectively. In particular, the *facts* $wm(p)$ and $att(p)$ which model that the wireless medium and the attacker know the pattern $p$, respectively. The fact *accept(s)* is used for signalling that the reply has been accepted by the source node. More precisely, the value of $s$ can be *accept(honest)* or *accept(advr)*. The derivation of the fact *accept(honest)* during the deduction algorithm means that the request and reply messages are forwarded only by honest nodes. The derivation of the fact *accept(advr)* means that the request and reply messages are forwarded by the attacker node who could modified them. The facts *badreq* and *badrep* are the special facts for signalling that the messages created by attacker include incorrect route. We will discuss them in detail later. Hence, the derivation of the facts *badreq* or *badrep* (could be both) and *accept(advr)* at the same time means that the source node accepts the route reply message containing an incorrect route. In addition, the fact $route(l_{src}^p, l_{dest}^p)$ is used to assign the source node $l_{src}^p$ and the target node $l_{dest}^p$ for a given route discovery. Finally, the fact $nbr(l_i^p, l_j^p)$ says that node $l_j^p$ is in the transmission range of node $l_i^p$.

The Horn-clauses use the following facts:

| $F ::=$ | *facts* |
|---|---|
| $wm(p)$ | *the wireless medium knows p* |
| $att(p)$ | *the compromised node knows p* |

| | |
|---|---|
| ***accept(s)*** | *special fact for signalling the "state" of the returned route* |
| $route(l^p_{src}, l^p_{dest})$ | $l^p_i$ *is source, and* $l^p_j$ *is a target* |
| *badreq, badrep* | *signalling that the attacker has created incorrect msg-es* |
| $nbr(l^p_i, l^p_j)$ | *node* $l^p_j$ *is a neighbor of node* $l^p_i$ |

*We stress that the predicates wm, route, nbr, the special facts badreq and badrep are new compared to the Proverif tool.*

## 9.3 Generating protocol clauses

*First, we lay down that the generation of Horn-clauses that presents the protocol is differ from the algorithm used in the Proverif tool.* 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:

$$P_i \stackrel{def}{=} !^{i_1} Init^{P_i} \mid !^{i_2} Interm^{P_i} \mid !^{i_3} Dest^{P_i}$$

Every node can start a route discovery towards any other node, in this case the *Init* 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:

$$\begin{aligned}
Init^{P_i} &\stackrel{def}{=} c_{P_i}(x_{dest}).\ Prot_{init}.\langle x_{msgreq}\rangle.!^{i_4} Rep^{P_i}_{init}.\\
Rep^{P_i}_{init} &\stackrel{def}{=} (x_{rep}).Verif_{init}.\langle ACCEPT\rangle.\\
Interm^{P_i} &\stackrel{def}{=} (y_{req}).Prot_{interm}.\langle y_{msgreq}\rangle.!^{i_5} Rep^{P_i}_{interm}.\\
Rep^{P_i}_{interm} &\stackrel{def}{=} (y_{rep}).Verif_{interm}.\langle y_{msgrep}\rangle.\\
Dest^{P_i} &\stackrel{def}{=} (z_{req}).Verif_{dest}.\langle z_{msgrep}\rangle.
\end{aligned}$$

Process $Init^{P_i}$ receives on the channel $c_{P_i}$ the ID of the destination node to which it starts the route discovery. Then $P_i$ goes through a protocol dependent processing part $Prot_{init}$, followed by broadcasting the request $x_{msgreq}$ and then it waits for the reply at the end.

On receiving a reply $x_{rep}$, process $Rep^{P_i}_{init}$ does verification steps in a protocol dependent manner and then signals the term $ACCEPT$ if $x_{rep}$ passes all verifications.

Process $Interm^{P_i}$ describes the case when $P_i$ is an intermediate node and says that when $P_i$ receives some request $y_{req}$ it goes through a protocol dependent processing part $Prot_{interm}$, and re-broadcasts the request $y_{msgreq}$. Finally, it waits for the reply.

On receiving a reply $y_{rep}$, process $Rep^{P_i}_{interm}$ does verification steps in a protocol dependent manner and then forwards the (may be modified) reply $y_{msgrep}$.

Process $Dest^{P_i}$ describes the case when $P_i$ is a destination node and says that when $P_i$ receives some request $z_{req}$ it does verification steps and then sends back the reply $z_{msgrep}$.

We assume that the reply messages $x_{rep}$, $y_{rep}$, $y_{msgrep}$, and $z_{msgrep}$ include the information of the addressee. Hence, when a node receives a reply it can check whether the reply is intended to it.

The exclamation mark $!^i$ models replication, where $i$ is a session identifier to distinguish different process instances in replication. Session identifier further is used to track attack traces.

Process $P_i$ is then translated to the protocol rules. A routing protocol modelled by $P_i$ is specified in the following clauses that serve as a template of the correct operation of the routing protocol:

(Protocol rules: Template of the correct operation) ::=
$R_1^{req}.\ route(x^p_{src}, x^p_{dest}) \longrightarrow wm((x^p_{msgreq}, honest))$
$R_1^{rep}.\ wm((x^p_{rep}, s)) \wedge nbr(x^p_{from}, l^p_{src}) \wedge Verif^{facts}_{init} \xrightarrow{p,i_2}$ **accept(s)**.
$R_2^{req}.\ wm((y^p_{req}, s)) \wedge nbr(y^p_{from}, y^p_{to}) \wedge Verif^{facts}_{intermreq} \xrightarrow{p,i_3} wm((y^p_{msgreq}, s))$.
$R_2^{rep}.\ wm((y^p_{rep}, s)) \wedge nbr(y^p_{from}, y^p_{to}) \wedge Verif^{facts}_{intermrep} \xrightarrow{p,i_4} wm((y^p_{msgrep}, s))$.
$R_3^{req}.\ wm((z^p_{req}, s)) \wedge nbr(z^p_{from}, l^p_{dest}) \wedge Verif^{facts}_{dest} \xrightarrow{p,i_5} wm((z^p_{msgrep}, s))$

*When a message is received by an attacker node:*
$R_1^{att}.\ wm((y^p_{req}, s)) \wedge nbr(y^p_{from}, l^p_{att}) \xrightarrow{p,i_6} att((y^p_{req}, s))$
$R_2^{att}.\ wm((y^p_{rep}, s)) \wedge nbr(y^p_{from}, l^p_{att}) \xrightarrow{p,i_7} att((y^p_{rep}, s))$
$R_3^{att}.\ wm((z^p_{req}, s)) \wedge nbr(z^p_{from}, l^p_{dest}) \wedge nbr(l^p_{dest}, l^p_{att}) \wedge Verif^{facts}_{dest} \xrightarrow{p,i_6} att((z^p_{msgrep}, s))$

We note that these rules represent the protocol independent form, however, in specified protocols such as SRP the translation will take into account the *Verif* and *ProtDep* parts. This may yield additional rules, and hypotheses. See Section 10.1 for examples. $Verif^{facts}_{init}$, $Verif^{facts}_{interm}$ and $Verif^{facts}_{dest}$ are set up by a conjunction of fact(s).

We assume that the route request and reply messages $x^p_{rep}$, $y^p_{req}$, $y^p_{rep}$ and $z^p_{req}$ include identifiers of the nodes from which they are received. The identifiers are specified by the patterns $x^p_{from}$, $y^p_{from}$, and $z^p_{from}$.

Rules $R_1^{req}$ and $R_1^{rep}$ model the operation of the source node. In particular, rule $R_1^{req}$ says that the source node $x^p_{src}$ generates the request message $x^p_{msgreq}$ and then broadcasts it. The initial value of $s$ is *honest*, which means that initially the request is broadcasted by a honest node. Rule $R_1^{rep}$ says that if the source node receives a reply message and if all verification steps the source node made on the reply pass then the returned route is accepted, this is modelled by the derivation of the fact **accept(s)**. The variable $s$ is a flag that takes its value from the set {*honest, advr*}, and is used to determine whether the attacker node intercepts a request/reply message (when $s$=*advr*) during the route discovery or not (when $s$=*honest*). More precisely, the derivation of the fact **accept(honest)** means that the request or reply are modified by honest nodes only, while when **accept(advr)** is derived the messages are manipulated by the attacker node.

The variable $s$ is appended after each request and reply messages forming the tuples $(x^p_{req}, s)$, $(y^p_{req}, s)$, $(y^p_{msgreq}, s)$, $(z^p_{req}, s)$, $(x^p_{rep}, s)$, $(y^p_{rep}, s)$, $(y^p_{msgrep}, s)$, and $(z^p_{msgrep}, honest)$.

Rules $R_2^{req}$ and $R_2^{rep}$ model the operation of intermediate nodes. In particular, rule $R_2^{req}$ says that if the node $y^p_{to}$ is in the transmission range of the $y^p_{from}$ then node $y^p_{to}$ receives the request $y^p_{req}$ sent by $y^p_{from}$. The value of $s$ is forwarded unchanged. The messages $y^p_{req}$ and $y^p_{msgreq}$ could be the same depending on the specified protocol. After processing the request $y^p_{req}$ node $y^p_{to}$ re-broadcasts it. Rule $R_2^{rep}$ says that if an intermediate node receives the message tuple $(y^p_{rep}, s)$ then it makes verification steps. If all verifications pass it appends $s$ unchanged to the reply $y^p_{msgrep}$ then forwards the message tuple $(y^p_{msgrep}, s)$. Again, the messages $y^p_{rep}$ and $y^p_{msgrep}$ could be the same depending on the specified protocol.

The rule $R_3^{req}$ models the operation of the destination node and says that if the destination node $l^p_{dest}$ is the neighbor of some honest intermediate node $z^p_{from}$ then the destination receives the request $z^p_{req}$. If all the verifications made by the destination pass it generates a reply $z^p_{msgrep}$ and adds the flag $s$ unchanged. Finally, it sends back the tuple $(z^p_{msgrep}, s)$.

Rules $R_1^{att}$ and $R_2^{att}$ concern the case when the attacker node $l^p_{att}$ intercepts the request $y^p_{req}$ and reply $y^p_{rep}$ because it is a neighbor of some honest node $y^p_{from}$. Rule $R_3^{att}$ concerns the case when the attacker node is a neighbor of the destination node but not a neighbor of node $y^p_{from}$, in addition, the destination node is a neighbor of node $y^p_{from}$. Rule $R_3^{att}$ says that the attacker node overhears the reply message sent back to $y^p_{from}$ by the destination node. $Verif^{facts}_{dest}$ is composed of a conjunction of facts and represents the verification steps made by the destination after receiving the request $z^p_{req}$ broadcasted by $y^p_{from}$.

The labelled arrow $\xrightarrow{p,i}$ is the implication that is labelled by the pair $(p, i)$, where $p$ is the broadcasted message by nodes $x^p_{from}$, $y^p_{from}$, $z^p_{from}$ and $i$ is the session identifier to identify that in which session the particular communication step happens. This pair is used for tracking the attack at the end when an attack is found by the algorithm. *We note that these protocol rules are novel compared with the ProVerif tool.*

## 9.4 Initial knowledge and computation ability of an attacker node

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

$(Init.\ knowl.) ::= \forall l^p_n$ neighbors of $l^p_{att}$:
$I_1.\ att(l^p_{att}),\ att(l^p_n);\ I_2.\ att(k(l_{att}, l^p_n));\ I_3.\ nbr(l_{att}, l^p_n).$

$I_1$ means that initially the attacker knows its own ID and the ID of its honest neighbors, $I_2$ means that the attacker possesses all the keys it shares with the honest nodes. Finally, $I_3$ means that the attacker is aware of its neighborhood.

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

(Computation ability - protocol independent) ::=
$C_1$. $att(i) \rightarrow att(n[i])$
$C_2$. *For each public function* $f$ *of n-arity*
$att(x_1^p) \wedge \ldots att(x_n^p) \rightarrow att(f(x_1^p, \ldots, x_n^p))$
$C_3$. *For each public function* $g$ *that*
$g(f(x_1^p, \ldots, x_n^p), y^p) \rightarrow x^p$:
$att(f(x_1^p, \ldots, x_n^p)) \wedge att(y^p) \rightarrow att(x^p)$
$C_4^1$. $att((y_{req}^p, s)) \wedge nbr(l_{att}^p, y_{to}^p) \wedge Verif_{att}^{facts} \xrightarrow{p,i_8} wm((y_{msgreq}^p, advr))$.
$C_4^2$. $att((y_{rep}^p, s)) \wedge nbr(l_{att}^p, y_{to}^p) \wedge Verif_{att}^{facts} \xrightarrow{p,i_9} wm((y_{msgrep}^p, advr))$.
$C_4^3$. $att((z_{req}^p, s)) \wedge nbr(l_{att}^p, l_{dest}^p) \wedge nbr(l_{dest}^p, l_{att}^p) \wedge Verif_{att}^{facts} \xrightarrow{p,i_{10}} att((z_{msgrep}^p, s))$.
$C_4^4$. $att((x_{rep}^p, s)) \wedge nbr(l_{att}^p, l_{src}^p) \wedge Verif_{att}^{facts} \xrightarrow{p,i_{11}}$ **accept(advr)**.

Rule $C_1$ means that the attacker node can create arbitrary new data $n$ such as fake identifiers, where $i$ is a session ID to identify that the data is created in which protocol run. 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.

The rules $C_4^1$, $C_4^2$, $C_4^3$ and $C_4^4$ say that the attacker can broadcast the messages it has. Rule $C_4^1$ describes the case in which the attacker broadcasts the request $y_{req}^p$ that is received by its neighbor $y_{to}^p$ who then outputs some message $y_{msgreq}^p$ if all verifications it made on $y_{req}^p$ (modelled by $Verif_{att}^{facts}$) pass. The value of the flag $s$ is *advr* signalling that the request is forwarded by the attacker. Rule $C_4^2$ says that the reply $y_{req}^p$ forwarded by the attacker is overheared by its neighbor $y_{to}^p$ who then forwards the reply with the flag *advr* signalling that the reply is forwarded by the attacker. Rule $C_4^3$ concerns the scenario when the destination node $l_{dest}^p$ and the attacker $l_{att}^p$ are neighbor of each other and says that if the request $z_{req}^p$ broadcasted by $l_{att}^p$ passes all the required verifications then the destination sends back a reply message along with the flag *honest* signalling that the reply is just generated and has not be seen by the attacker. Finally, rule $C_4^4$ concerns the scenario when the source node $l_{src}^p$ is a neighbor of the attacker $l_{att}^p$ and says that if the reply $z_{rep}^p$ forwarded by $l_{att}^p$ passes all the required verifications made by $l_{src}^p$ then the reply is accepted. The implication $\xrightarrow{p,i}$ includes the message and session ID that required for tracking attacks.

*We note that the* $I_1$, $I_3$ *of the initial knowledge, and the last four rules of the computation ability are novel compared with the ProVerif tool.*

## 9.5 Deductive algorithm

The operation of the protocol is modelled by rerolution steps, defined as the following:

**Definition 12.** *Given two rules $r_1$=$H_1 \rightarrow C_1$, and $r_2$= $F \wedge H_2 \rightarrow C_2$, where $F$ is **any** hypotheses of $r_2$, and $F$ is unifiable with $C_1$ with the most general unifier $\sigma$, then the resolution $r_1 \circ_F r_2$ of them yields a new rule $H_1\sigma \wedge H_2\sigma \rightarrow C_2\sigma$.*

Here $H_1$ and $H_2$ hypotheseses are multiset of facts, which means that the order of the facts in the hypotheses are irrelevant. For instance, if $r_1$=$route(l_1^p, l_2^p)$ and $r_2$=$route(x_{src}, x_{dest}) \xrightarrow{p,i_1} wm(x^p_{msgreq})$ then $r_1 \circ_F r_2$= $wm(x^p_{msgreq})\sigma$, where $F$=$route(x_{src}, x_{dest})$, and $\sigma = \{x_{src} \leftarrow l_1^p, x_{dest} \leftarrow l_2^p\}$.

In addition, let us recall the example in Figure 2. The resolution of $route(l_1^p, l_3^p)$ and the rule $R_1^{req}$ with $p$=(*req, $x_{src}$, $x_{dest}$, ID, MAC*) yields the fact $wm$(*req,* $l_1^p$, $l_3^p$, *ID, MAC*) with the unifier $\{x^p_{src} \leftarrow l_1^p, x^p_{dest} \leftarrow l_3^p\}$, and $F$= $route(x^p_{src}, x^p_{dest})$. This resolution step models the broadcasting of the route request message generated by the node $l_1^p$.

The algorithm composed of two phases. In the first phase the route request message wanders from the source towards the attacker node. At first, the attacker node owns only its initial knowledge and stays in the idle state waiting for route request messages. After the attacker receives a message, based on the available information and its computation ability it tries to generate messages that includes an incorrect route and are accepted by at least one honest neighbor. Formally this means that the created message by the attacker is unifiable with the hypotheses of rules $C_4^1$, $C_4^2$, $C_4^3$ and $C_4^4$.

**Algorithm 1**: Main()

**Inputs**: $\mathcal{T}_0 = \mathcal{T}_0^{req} \cup \mathcal{T}_0^{rep}$, $\{\mathcal{N}_{l_i^p}\}$, $\{\mathcal{N}_{l^p_{att}}\}$, $route(l^p_{src}, l^p_{dest})$, $\mathcal{K}$, $\mathcal{A}$.
$H$ = empty; $H_{att}$ = empty; $\mathcal{N}_{visited} = \{nbr(x^p, l^p_{src})\}$;
1. $P$ := $route(l^p_{src}, l^p_{dest}) \circ_{route(x^p_{src}, x^p_{dest})} R_1^{req}$,
*where* $\sigma = \{x^p_{src} \leftarrow l^p_{src}, x^p_{dest} \leftarrow l^p_{dest}\}$;
2. $H'$ := add $\sigma R_1^{rep}$ to $H$; $\mathcal{T}_0^{req} \cup \{R_3^{req} \leftarrow \sigma R_3^{req}\}$;
**call** function $resolution_1(P, R_2^{req}, H')$;
**call** function $resolution_2(P, R_1^{att}, H')$;

Function **resolution**$_1(P, R_2^{req}, H)$

1. $Temp := P \circ_{wm((y^p_{req},s))} R_2^{req}$, where the unifier is some $\sigma^1$;
   $loc := \sigma^1 y^p_{from}$;
2. **for** $\forall$ fact $F_j \in \mathcal{N}_{loc}$ **do**
   $P_j := F_j \circ_{nbr(loc,y^p_{to})} Temp$ with some unifier $\sigma_j^2$;
   **for** $\forall$ fact $f_k \in \mathcal{N}_{visited}$ **do**
   **if** $\sigma_j^2 nbr(loc, y^p_{to})$ and $f_k$ are unifiable then choose next $F_j$.;
   **endfor**
   **if** $Q_j := res(P_j, Verif^{facts}_{intermreq})$ succeeds with some unifier $\tilde{\sigma}$ **then**{
   $H' :=$ add $\sigma^1\sigma_j^2\tilde{\sigma}R_2^{rep}$ to the end of $H$;
   $\mathcal{N}_{visited} := \mathcal{N}_{visited} \cup \{\sigma_j^2 nbr(x^p, y^p_{to})\}$;
   **call** function $resolution_1(Q_j, R_2^{req}, H')$;
   **call** function $resolution_2(Q_j, R_1^{att}, H')$;
   **call** function $resolution_3(Q_j, R_3^{req}, H')$;
   **call** function $resolution_4(Q_j, R_3^{att})$;
   }
   **endfor**

Function **resolution**$_2(P, R_1^{att}, H)$

1. $Temp := P \circ_{wm((y^p_{req},s))} R_1^{att}$, where the unifier is some $\sigma^1$;
   $loc := \sigma^1 y^p_{from}$;
2. **if** $nbr(loc, l^p_{att}) \in \mathcal{N}_{loc}$ **then** {
   $P := nbr(loc, l^p_{att}) \circ_{nbr(loc,l^p_{att})} Temp$, with $\{loc \leftarrow loc, l^p_{att} \leftarrow l^p_{att}\}$;
   $\mathcal{K} := \mathcal{K} \cup \{P\}$; Add $H$ to the end of $H_{att}$;
   **call** function $Attacker_{req}(\mathcal{K}, \mathcal{A}, H)$;
   }

Function **resolution**$_3(P, R_3^{req}, H)$

1. $Temp := P \circ_{wm((z^p_{req},s))} R_3^{req}$, where the unifier is some $\sigma^1$;
   $loc := \sigma^1 z^p_{from}$;
2. **if** $nbr(loc, l^p_{dest}) \in \mathcal{N}_{loc}$ **then** {
   $P := nbr(loc, l^p_{dest}) \circ_{nbr(loc,l^p_{dest})} Temp$, with $\{loc \leftarrow loc, l^p_{dest} \leftarrow l^p_{dest}\}$;
   **if** $Q := res(P, Verif^{facts}_{dest})$ succeeds with some unifier $\tilde{\sigma}$ **then**{
   **call** function $resolution_5(Q, H)$;
   }
   }

Function **resolution**$_4(P, R_3^{att})$

1. $Temp := P \circ_{wm((z^p_{req},s))} R_3^{att}$, where the unifier is some $\sigma^1$;
   $loc := \sigma^1 z^p_{from}$;
2. **if** $nbr(loc, l^p_{dest}) \in \mathcal{N}_{loc}$ **then** {
   /* $l^p_{dest} = \sigma_1 x^p_{dest}$, where $\sigma_1$ is in Main(). */
   $P := nbr(loc, l^p_{dest}) \circ_{nbr(loc, l^p_{dest})} Temp$, with $\{loc \leftarrow loc,\ l^p_{dest} \leftarrow l^p_{dest}\}$;
   **if** $nbr(l^p_{dest}, l^p_{att}) \in \mathcal{N}_{l^p_{dest}}$ **and**
   $Q := nbr(l^p_{dest}, l^p_{att}) \circ_{nbr(l^p_{dest}, l^p_{att})} P$, with $\{l^p_{dest} \leftarrow l^p_{dest},\ l^p_{att} \leftarrow l^p_{att}\}$ **and**
   $F := res(Q, Verif^{facts}_{dest})$ succeeds with some unifier $\tilde{\sigma}$ **then**{
   $\mathcal{K} := \mathcal{K} \cup \{F\}$;
   **call** function $Attacker_{rep}(\mathcal{K}, \mathcal{A})$;
   }
  }

Function **resolution**$_5(P, H)$

$h :=$ last element of $H$;
**if** $h$ is an instance of $R_2^{rep}$ **then** {
1. $Temp := P \circ_{wm(\sigma_{close}(y^p_{rep},s))} h$, where the unifier is some $\sigma^1$;
2. **if** $nbr(\sigma_{close} y^p_{from}, \sigma_{close} y^p_{to}) \in \mathcal{N}_{\sigma_{close} y^p_{from}}$ **then** {
   $P := nbr(\sigma_{close} y^p_{from}, \sigma_{close} y^p_{to}) \circ_{nbr(\sigma_{close} y^p_{from}, \sigma_{close} y^p_{to})} Temp$,
   with $\{\sigma_{close} y^p_{from} \leftarrow \sigma_{close} y^p_{from},\ \sigma_{close} y^p_{to} \leftarrow \sigma_{close} y^p_{to}\}$;
   **if** $Q := res(P, Verif^{facts}_{intermrep})$ succeeds with some unifier $\tilde{\sigma}$ **then**{
   $H' :=$ remove $h$ from the end of $H$;
   **call** function $resolution_5(Q, H')$;
   **call** function $resolution_6(Q, R_2^{att})$;
   }
  }
} **else**
**if** $h$ is an instance of $R_1^{rep}$ **then** {
3. $Temp := P \circ_{wm(\sigma_{close}(x^p_{rep},s))} h$, where the unifier is some $\sigma^1$;
4. **if** $nbr(\sigma_{close} y^p_{from},\ l^p_{src}) \in \mathcal{N}_{\sigma_{close} y^p_{from}}$ **then** {
   $P := nbr(\sigma_{close} y^p_{from},\ l^p_{src}) \circ_{nbr(\sigma_{close} y^p_{from},\ l^p_{src})} Temp$,
   with $\{\sigma_{close} y^p_{from} \leftarrow \sigma_{close} y^p_{from},\ l^p_{src} \leftarrow l^p_{src}\}$;
   **if** $Q := res(P, Verif^{facts}_{init})$ succeeds with some unifier $\tilde{\sigma}$ **then**{
   **return** $Q$;
   }
  }
}

Function $\mathbf{resolution}_6(P, R_2^{att})$

1. $Temp := P \circ_{wm(\sigma_{closed}(y^p_{rep},s))} R_2^{att}$, where the unifier is some $\sigma^1$;
2. **if** $nbr(\sigma_{close}y^p_{from},\ l^p_{att}) \in \mathcal{N}_{\sigma_{close}y^p_{from}}$ **then** {
   $P := nbr(\sigma_{close}y^p_{from},\ l^p_{att}) \circ_{nbr(\sigma_{close}y^p_{from},\ l^p_{att})}$ *Temp*,
   with $\{\sigma_{close}y^p_{from} \leftarrow \sigma_{close}y^p_{from},\ l^p_{att} \leftarrow l^p_{att}\}$;
   $\mathcal{K} := \mathcal{K} \cup \{P\}$; **call** function $Attacker_{rep}(\mathcal{K}, \mathcal{A})$;
   }

Function $Attacker_{req}(\mathcal{K}, \mathcal{A}, H)$

$badReq := computeBadReq(\mathcal{K}, \{C_1, C_2, C_3\})$, where the unifier is some $\sigma^1$;
**call** function $\mathbf{resolution}_7(badReq, C_4^1, H)$;
**call** function $\mathbf{resolution}_8(badReq, C_4^3, H)$;

Function $\mathbf{resolution}_7(badReq, C_4^1, H)$

1. $Temp := badReq \circ_{att((y^p_{req},s))} C_4^1$, where the unifier is some $\sigma^1$;
2. **for** $\forall$ fact $F_j \in \mathcal{N}_{l^p_{att}}$ **do**
   $P_j := F_j \circ_{nbr(l^p_{att},y^p_{to})}$ *Temp* with some unifier $\sigma_j^2$;
   **for** $\forall$ fact $f_k \in \mathcal{N}_{visited}$ **do**
   **if** $\sigma_j^2 nbr(l^p_{att}, y^p_{to})$ and $f_k$ are unifiable then choose next $F_j$;
   **endfor**
   **if** $Q_j := res(P_j, Verif_{att}^{facts})$ succeeds with some unifier $\tilde{\sigma}$ **then**{
   $\mathcal{N}_{visited} := \mathcal{N}_{visited} \cup \{\sigma_j^2 nbr(x^p, y^p_{to})\}$;
   **call** function $resolution_1(Q_j, R_2^{req}, H)$;
   **call** function $resolution_3(Q_j, R_3^{req}, H)$;
   }
   **endfor**

Function $\mathbf{resolution}_8(badReq, C_4^3, H)$

1. $Temp := badReq \circ_{att((z^p_{req},s))} C_4^3$, where the unifier is some $\sigma^1$;
2. **if** $nbr(l^p_{att}, l^p_{dest}) \in \mathcal{N}_{l^p_{att}}$ **then** {
   $P := nbr(l^p_{att}, l^p_{dest}) \circ_{nbr(l^p_{att},l^p_{dest})}$ *Temp*, with $\{l^p_{att} \leftarrow l^p_{att},\ l^p_{dest} \leftarrow l^p_{dest}\}$;
   **if** $nbr(l^p_{dest}, l^p_{att}) \in \mathcal{N}_{l^p_{dest}}$ **and**
   $Q := nbr(l^p_{dest}, l^p_{att}) \circ_{nbr(l^p_{dest},l^p_{att})} P$, with $\{l^p_{dest} \leftarrow l^p_{dest},\ l^p_{att} \leftarrow l^p_{att}\}$ **and**
   $F := res(Q, Verif_{att}^{facts})$ succeeds with some unifier $\tilde{\sigma}$ **then**{
   $\mathcal{K} := \mathcal{K} \cup \{F\}$;
   **call** function $Attacker_{rep}(\mathcal{K}, \mathcal{A})$;
   }
   }

Function $Attacker_{rep}(\mathcal{K}, \mathcal{A})$

$badRep := computeBadRep(\mathcal{K}, \{C_1, C_2, C_3\})$, where the unifier is some $\sigma^1$;
**call** function $\mathbf{resolution}_9(badRep, C_4^2)$;
**call** function $\mathbf{resolution}_{10}(badRep, C_4^4)$;

Function **resolution**$_9(badRep, C_4^2)$

**for** $\forall H_i \in H_{att}$ **do**
$h_i$ := last element of $H_i$;
1. $Temp := badRep \circ_{att(\sigma_{close}(y^p_{rep},s))} h_i$, where the unifier is some $\sigma^1$;
2. **if** $nbr(l^p_{att},\ \sigma_{close}y^p_{to}) \in \mathcal{N}_{l^p_{att}}$ **then** {
$P := nbr(l^p_{att},\ \sigma_{close}y^p_{to}) \circ_{nbr(l^p_{att},\ \sigma_{close}y^p_{to})} Temp$,
with $\{l^p_{att} \leftarrow l^p_{att}, \sigma_{close}y^p_{to} \leftarrow \sigma_{close}y^p_{to}\}$;
**if** $Q := res(P, Verif^{facts}_{att})$ succeeds with some unifier $\tilde{\sigma}$ **then**{
$H'_i$ := remove $h_i$ from the end of $H_i$;
**call** function $resolution_5(Q, H'_i)$;
}
}
**endfor**

Function **resolution**$_{10}(badRep, C_4^4)$

$h_i$ := 1st element of $H_i$ for some $H_i \in H_{att}$; /* $h_i$ is an instance of $R_1^{rep}$ */
1. $Temp := P \circ_{att(\sigma_{close}(x^p_{rep},s))} h_i$, where the unifier is some $\sigma^1$;
2. **if** $nbr(l^p_{att},\ l^p_{src}) \in \mathcal{N}_{l^p_{att}}$ **then** {
$P := nbr(l^p_{att},\ l^p_{src}) \circ_{nbr(l^p_{att},\ l^p_{src})} Temp$,
with $\{l^p_{att} \leftarrow l^p_{att}, l^p_{src} \leftarrow l^p_{src}\}$;
**if** $Q := res(P, Verif^{facts}_{att})$ succeeds with some unifier $\tilde{\sigma}$ **then**{
**return** $Q$;
}
}

**Algorithm 1.** *Intuitively, the algorithm uses breadth-first search to reach the destination node $l^p_{dest}$ from the source node $l^p_{src}$. Each node broadcasts its message to neighbors. This is modelled by two resolution steps at each intermediate node. Next we will discuss each step of the algorithm for better understanding purpose:*

*The input of the algorithm is the tuple ($\mathcal{T}_0$, $\{\mathcal{N}_{l^p_i}\}$, $\{\mathcal{N}_{l^p_{att}}\}$, $route(l^p_{src}, l^p_{dest})$, $\mathcal{K}$, $\mathcal{A}$) : $\mathcal{T}_0$ is the set of protocol rules; the two sets $\{\mathcal{N}_{l^p_i}\}$ and $\{\mathcal{N}_{l^p_{att}}\}$ specify the neighborhood of the honest node $l^p_i$ and the attacker node $l^p_{att}$ which are given by the set of facts $nbr(l^p_i, l^p_j)$ and $nbr(l^p_{att}, l^p_j)$, respectively; the fact $route(l^p_{src}, l^p_{dest})$ specifies the source and target nodes for a given route discovery; $\mathcal{A}$ and $\mathcal{K}$ are the sets of attacker computation rules and the knowledge base of the attacker, respectively.*

*$H$ is a **record** of rules that is used to store the **expected reply messages** of each node on a given route, which correspond to the request it has broadcasted or forwarded. Formally, these reply can be obtained by updating the proper rules $R_1^{rep}$ and $R_2^{rep}$. The update is done by applying the unifier of the resolutions involving the corresponding "request" rules $R_1^{req}$ and $R_2^{req}$ to the "reply" rules $R_1^{rep}$ and $R_2^{rep}$. For instance, let us recall $route(l^p_1, l^p_3) \circ_{route(x^p_{src}, x^p_{dest})} R_1^{req}$ where the unifier $\sigma$ is $\{x^p_{src} \leftarrow l^p_1,\ x^p_{dest} \leftarrow l^p_3\}$. This resolution models the broadcasting of the initial request by the source node. After this resolution step the corresponding*

*"reply" rule of $R_1^{req}$, $R_1^{rep}$ is updated by applying $\sigma$ to $R_1^{rep}$: $\sigma R_1^{rep}$. Intuitively, this means that after broadcasting the route request the source node $l_1^p$ waits for the corresponding reply. $H_{att}$ is a record of records $Hs$ and is used to store the reply of each node lying on the routes between the source node and the attacker node that not contains the destination node.*

*The sets $\mathcal{S}_1$, $\mathcal{S}$ and $\mathcal{S}'$ are used to store the facts resulted from resolution steps; $Temp_i$ is used to store the rules resulted from resolution steps; $loc_i$ is used to store node IDs; set $\mathcal{N}_{visited}$ is used to store the facts $nbr(l_i^p, l_j^p)$ in which node $l_j^p$ has already accepted a route request, so that it will drops the same copies of the request it receives later.*

*At the begining, $H$ is empty; $H_{att}$ is empty; $\mathcal{N}_{visited} = \{nbr(x^p, l_{src}^p)\}$, that is, at first the source node is marked as visited so that after it broadcasts the initial request it will drop any further copy of this request; $\mathcal{T}_0 = \{R_1^{req}, R_2^{req}, R_1^{rep}, R_2^{rep}, R_3^{req}, R_1^{att}, R_2^{att}, R_3^{att}\}$. This set of rules is then organized into two disjunct sets $\mathcal{T}_0^{req}$ that models the receiving and broadcasting of route request message: $\{R_1^{req}, R_2^{req}, R_3^{req}, R_1^{att}, R_3^{att}\}$, and the $\mathcal{T}_0^{rep}$ which models the phase of waiting for route reply message: $\{R_1^{rep}, R_2^{rep}, R_2^{att}\}$.*

*The algorithm ends when no more resolution steps can be executed. This can happen when*

- *When the route discovery phase succeeded: the facts in $\mathcal{T}^{rep}$ do not contain any variable, so that fixpoint is reached and no further resolution step can be made.*
- *When the route discovery phase fails (there is no route between the source and the destination nodes) the fixpoint is reached, and no more change can be done on the set $\mathcal{T}^{rep}$.*

1. *Funtion Main(): As the first step, the algorithm computes the resolution $P := route(l_{src}^p, l_{dest}^p) \circ_{route(x_{src}^p, x_{dest}^p)} R_1^{req}$, where $R_1^{req} = route(x_{src}^p, x_{dest}^p) \longrightarrow wm((x_{msgreq}^p, honest))$. The resolution is successful with the unifier $\sigma$: $\{x_{src}^p \leftarrow l_{src}^p, x_{dest}^p \leftarrow l_{dest}^p\}$. As a result we have $P = wm((\sigma x_{msgreq}^p, honest))$. Intuitively, this means that the source node $l_{src}^p$ starts the route discovery towards node $l_{dest}^p$ by creating the initial route request message and broadcasts it. Figure 11 illustrates this step. After the resolution happens the unifier $\sigma$ is also applied to $R_1^{rep}$ because it belongs to the same process as $R_1^{req}$. Hence, the updated rule $R_1^{rep}$ is added to the record $H$; and the rule $R_3^{req}$ is also updated by bounding the ID of the source and destination node in it: $\mathcal{T}_0^{req} := \mathcal{T}_0^{req} \cup \{R_3^{req} \leftarrow \sigma_1 R_3^{req}\}$, that is, $R_3^{req}$ is replaced by $\sigma_1 R_3^{req}$ in $\mathcal{T}_0^{req}$. Finally, the functions $resolution_1(P, R_2^{req}, H')$ and $resolution_2(P, R_1^{att}, H')$ are called. Function $resolution_1(P, R_2^{req}, H')$ describes the propagation of the request between intermediate nodes, while $resolution_2(P, R_1^{att}, H')$ describes the scenario in which the attacker node intercepts a request.*
2. *In the second step, after receiving the request the neighbors of the source node $l_{src}^p$ process the request and re-broadcast it. This is modelled by the*

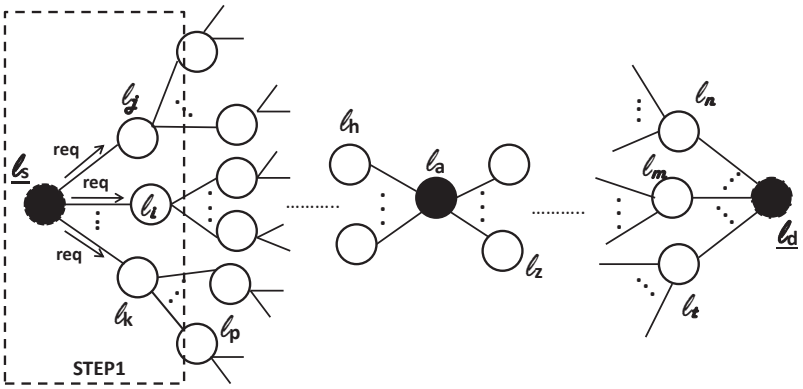


Figure 11: The scenario in which the source node broadcasts the initial request message. In this scenario the ID of the source node and destination node are $l^p_{src}$ and $l^p_{dest}$, respectively. The ID of the attacker node is $l^p_{att}$.

*functions* $\underline{resolution_1(P, R_2^{req}, H)}$ *and* $\underline{resolution_2(P, R_1^{att}, H)}$. *The first function represents the case when a honest node receives the request broadcasted by the source node, while the second function represents the case when the attacker node receives the request broadcasted by the source node.*

*(i). In the function* $resolution_1(P, R_2^{req}, H)$, *first the resolution* $P \circ_{wm((y^p_{req}, s))} R_2^{req}$ *is computed. The unifier* $\sigma^1$ *of this resolution is* $\{y^p_{from} \leftarrow l^p_{src}, y^p_{req} \leftarrow \sigma x^p_{msgreq}, s \leftarrow honest\}$. *Then, loc is computed by binding the variable* $y^p_{from}$ *with a node ID* $l^p_{src}$. *Formally, this means that the substitution* $\sigma^1$ *is applied to* $y^p_{from}$: $\sigma^1 y^p_{from}$. *We assume that each resolution step between honest nodes is successful because honest nodes follow the protocol and send correct messages. Thus, we have:*

$$Temp = nbr(l^p_{src}, y^p_{to}) \wedge Verif^{facts}_{intermreq} \xrightarrow{p,i_3} wm((\sigma^1 y^p_{msgreq}, honest)),$$

*the first fact* $wm((\sigma_1 x^p_{msgreq}, honest))$ *on the left side is eliminated from* $R_2^{req}$. *(Figure 12 illustrates this step.)*

*Then, the resolution of Temp and all the facts in the set* $N_{l^p_{src}}$, *which specify all the neighbors of the source node* $l^p_{src}$, *is computed. This models the receiving of the request by all the neighbors of* $l^p_{src}$. *We note that at this point* $\sigma^1$ *is* $\{y^p_{from} \leftarrow l^p_{src}\}$, *and loc is* $l^p_{src}$.

$$\forall \text{ fact } F_j \in \mathcal{N}_{l^p_{src}} : P_j := F_j \circ_{nbr(l^p_{src}, y^p_{to})} Temp,$$

*where the unifiers* $\sigma^2_j$ *of these resolutions are* $\{y^p_{to} \leftarrow l^p_j\}$, $\forall l^p_j \in N_{l^p_{src}}$. *Hence, we have* $|N_{l^p_{src}}|$ *number of new facts:*

$$\forall l_j^p \in N_{l_{src}^p}\colon P_j = Verif_{intermreq}^{facts} \xrightarrow{p,i_3} wm(\sigma^1\sigma_j^2(y_{msgreq}^p, honest)),$$

*because* $nbr(l_{src}^p, l_j^p)$ *is eliminated from Temp as the result of the resolutions. In the next* ***for*** *cycle, the algorithm checks if the node* $y_{to}^p$ *has already received the request: At this point, we have* $\sigma_j^2 nbr(loc, y_{to}^p) = nbr(l_{src}^p, \sigma_j^2 y_{to}^p)$ *and* $f_k = nbr(x^p, l_{src}^p)$*. These two facts are not unifiable due to* $l_{src}^p$ *cannot be unified with* $\sigma_j^2 y_{to}^p$*. As the next step the algorithm makes the resolution steps using* $P_j$ *and the facts in* $Verif_{intermreq}^{facts}$ $(res(P_j, Verif_{intermreq}^{facts}))$*. If the resolution steps made by the algorithm are all successful with the unifier* $\tilde{\sigma}$ *then the resulted fact* $Q_j$ *is:*

$$Q_j = wm((\sigma^1\sigma_j^2 y_{msgreq}^p, honest)).$$

*This intuitively means that node* $l_j^p$ *has processed (and possibly has modified) and re-broadcasts the request message* $\sigma^1\sigma_j^2 y_{msgreq}^p$ *embedded in the tuple* $(\sigma^1\sigma_j^2 y_{msgreq}^p, honest)$*. Afterwards, node* $y_{to}^p$ *is marked as visited, that is,* $\mathcal{N}_{visited} := \mathcal{N}_{visited} \cup \{nbr(x^p, y_{to}^p)\}$*.*

*Finally, the updated instance of* $R_2^{rep}$*,* $\sigma^1\sigma_j^2\tilde{\sigma}R_2^{rep}$ *are added to* $H$*, and the four functions are called with the resulted fact* $Q_j$ *and updated* $H$*.*

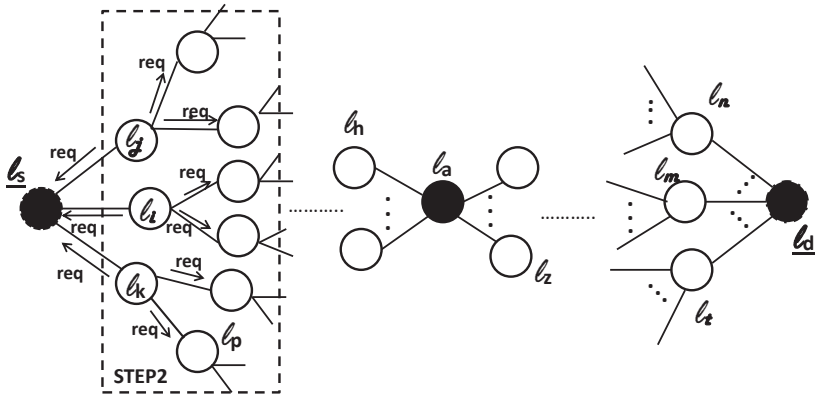


Figure 12: This figure illustrates the case when the neighbors of the source node $l_s^p$ process the received request and broadcast it.

*(ii). In the function* $resolution_2(P, R_1^{att}, H)$*, first* $P \circ_{wm((y_{req}^p, s))} R_1^{att}$ *is computed. As the result we have*

$$Temp = nbr(loc, l_{att}^p) \xrightarrow{p,i_6} att((\sigma^1 y_{req}^p, honest)).$$

*Afterwards, the algorithm checks if* $nbr(loc, l_{att}^p) \in \mathcal{N}_{loc}$*, that is, the attacker node is a neighbor of node loc. If so then the resolution* $nbr(loc, l_{att}^p)$ $\circ_{nbr(loc, l_{att}^p)}$ *Temp is computed, with the unifier* $\{loc \leftarrow loc,\ l_{att}^p \leftarrow l_{att}^p\}$*,*

*which is an "identity"'. As the result we have* $P = att((\sigma^1 y^p_{req}, honest))$, *which intuitively means that the attacker node has intercepted the request* $\sigma^1 y^p_{req}$ *broadcasted by node loc. The knowledge base of the attacker is impoved by* $P$, *that is,* $\mathcal{K} := \mathcal{K} \cup \{P\}$; *and the record* $H$ *is added to* $H_{att}$. *Finally, the function* $Attacker_{req}(\mathcal{K}, \mathcal{A}, H)$ *is called, which describe the behavior of the attacker node after intercepting a request.*

3. *The behavior of the further intermediate nodes (which are not a neighbor of the source* $l^p_{src}$*) is similar to the point 2 above. The only difference is that now the destination node can be reached. This scenario happens when the function* $\underline{resolution_3(P, R_3^{req}, H)}$ *is called.*

   *In the function* $resolution_3(P, R_3^{req}, H)$, *first the resolution* $P \circ_{wm((z^p_{req}, s))} R_3^{req}$ *is computed which yields*

$$Temp = nbr(loc, l^p_{dest}) \wedge Verif^{facts}_{dest} \xrightarrow{p, i_5} wm((\sigma^1 z^p_{msgrep}, \sigma^1 s)).$$

   *Then the algorithm checks whether the destination node* $l^p_{dest}$ *is a neighbor of node loc. If so then the destination node makes verification steps on the request. These are described by the series of resolutions*

$$P := nbr(loc, l^p_{dest}) \circ_{nbr(loc, l^p_{dest})} Temp, \text{ if } nbr(loc, l^p_{dest}) \in \mathcal{N}_{loc}.$$

   *with the unifier* $\{loc \leftarrow loc,\ l^p_{dest} \leftarrow l^p_{dest}\}$. *As the result we have* $P = wm((\sigma^1 z^p_{msgrep}, \sigma^1 s))$, *which intuitively means that in case all the verification that the destination made on the request are ok then it generates the reply message and sends it back to the node from which it received the request. Thereafter, a series of resolutions that includes the resulted rule/fact* $P$ *and the facts in* $Verif^{facts}_{dest}$ *is computed, which is defined by the function* $res(P, Verif^{facts}_{dest})$. *Intuitively,* $res(P, Verif^{facts}_{dest})$ *describes the verification that the destination node makes on the received request. The fact* $Q$ *is the result of this series of resoultions. At the end, the function* $resolution_5(Q, H)$ *is called, which specifies how the reply propagates backward. The record* $H$ *includes the rules/facts that model the reply messages which the nodes in the route are waiting for. We note that the reply sent back by the destination node is overheared by the nodes which are within the its transmission range. However, we focus only on the case when the attacker node intercepts the reply, because the honest nodes will drop the replies which are not intended to it.*

4. *The function* $\underline{resolution_4(P, R_3^{att})}$ *specifies the scenario in which the attacker node overhears the reply sent by the destination. First, the resolution* $P \circ_{wm((z^p_{req}, s))} R_3^{att}$ *is computed, which yields*

$$Temp := nbr(z^p_{from}, l^p_{dest}) \wedge nbr(l^p_{dest}, l^p_{att}) \wedge Verif^{facts}_{dest} \xrightarrow{p, i_6} att((z^p_{msgrep}, s)).$$

*with the unifier* $\sigma^1$*. Afterwards, the algorithm checks if the destination node* $l^p_{dest}$ *is a neighbor of node loc that broadcasted the request, then it checks if the attacker node* $l^p_{att}$ *is a neighbor of the destination node* $l^p_{dest}$ *that sent the reply. These are desribed by the two resolutions:*

$$P := nbr(loc, l^p_{dest}) \circ_{nbr(loc,l^p_{dest})} \textit{Temp, if } nbr(loc, l^p_{dest}) \in \mathcal{N}_{loc}, \textit{ and}$$

$$Q := nbr(l^p_{dest}, l^p_{att}) \circ_{nbr(l^p_{dest},l^p_{att})} P, \textit{ if } nbr(l^p_{dest}, l^p_{att}) \in \mathcal{N}_{l^p_{dest}}.$$

*Afterwards,* $res(Q, Verif^{facts}_{dest})$ *is called, which represents the verification made by the destination on the request it received from node loc. If all resolution steps (i.e, verification step) are successful the resulted fact* $F$ *represents the reply sent by the destination.* $F$ *is added to the knowledge base of the attacker:* $\mathcal{K} := \mathcal{K} \cup \{F\}$*. Finally, the function* $Attacker_{rep}(\mathcal{K}, \mathcal{A})$ *which describes the behaviour of the attacker node after overhearing the reply.*

5. *In* $\underline{resolution_5(P, H)}$*, first the last element* $h$ *of* $H$ *is examined if it is an instance of* $R^{rep}_2$ *or* $R^{rep}_1$*. If* $h$ *is is an instance of* $R^{rep}_2$ *(i.e.,* $h = \sigma_{close} R^{rep}_2$*), which means that the previous node in the route is an intermediate node, first* $P \circ_{wm(\sigma_{close}(y^p_{rep},s))} h$ *is computed. Here* $wm(\sigma_{close}(y^p_{rep}, s))$ *is a fact in the hypotheses of* $h$*. The substitution* $\sigma_{close}$ *is applied to* $(y^p_{rep}, s)$ *which represents that some variables in* $(y^p_{rep}, s)$ *is bound to a constant value during the resolutions maded in the request phase. The resolution* $P \circ_{wm(\sigma_{close}(y^p_{rep},s))} h$ *yields*

$$Temp := nbr(\sigma_{close} y^p_{from}, \sigma_{close} y^p_{to}) \wedge \sigma_{close} Verif^{facts}_{intermrep} \xrightarrow{p,i_4} wm((\sigma_{close} y^p_{msgrep}, \sigma_{close} s)).$$

*Thereafter, it checks if node* $\sigma_{close} y^p_{to}$ *is a neighbor of node* $\sigma_{close} y^p_{to}$*. If so then* $res(P, Verif^{facts}_{intermrep})$ *is called, which returns* $Q$ *if all resolutions are successful. Then* $h$ *is removed from the end of record* $H$ *yielding* $H'$*. Finally, the function* $resolution_5(Q, H')$ *is recursively called, and* $resolution_6(Q, R^{att}_2)$ *is called as well.*

*If* $h$ *is is an instance of* $R^{rep}_1$ *(i.e.,* $h = \sigma_{close} R^{rep}_1$*), which means that the previous node in the route is the source node, first* $P \circ_{wm(\sigma_{close}(x^p_{rep},s))} h$ *is computed, which yields*

$$Temp := nbr(\sigma_{close} x^p_{from}, l^p_{src}) \wedge Verif^{facts}_{init} \xrightarrow{p,i_2} \textbf{\textit{accept}}(\sigma s).$$

*Here* $\sigma$ *is the substitution that binds* $s$ *to* ***honest*** *or* ***advr*** *depends on the previous resolution steps. Afterwards, the algorithm checks if the destination node is a neighbor of node* $\sigma_{close} x^p_{from}$ *which forwards the reply.*

$$P := nbr(\sigma_{close} y^p_{from},\ l^p_{src}) \circ_{nbr(\sigma_{close} y^p_{from},\ l^p_{src})} \mathit{Temp},\ \mathit{if}\ nbr(\sigma_{close} y^p_{from},\ l^p_{src}) \in \mathcal{N}_{\sigma_{close} y^p_{from}}.$$

*If so then* $res(P, Verif^{facts}_{init})$ *is called, which returns* $\boldsymbol{accept}(\sigma s)$ *if all the verification steps in* $res(P, Verif^{facts}_{init})$ *are successful.*

6. *The function* $\underline{resolution_6(P, R^{att}_2)}$ *represents the scenario in which the attacker overhears the reply message forwarded by intermediate nodes. First the resolution* $P \circ_{wm(\sigma_{closed}(y^p_{rep}, s))} R^{att}_2$ *is computed, which yields*

$$\mathit{Temp} := nbr(\sigma_{closed} y^p_{from}, l^p_{att}) \xrightarrow{p, i_7} att((\sigma_{closed} y^p_{rep}, \sigma_{closed} s)).$$

*After that, the algorithm checks if the attacker node* $l^p_{att}$ *is within the transmission range of node* $\sigma_{closed} y^p_{from}$ *which forwards the reply. This is described by the following resolution*

$$P := nbr(\sigma_{close} y^p_{from},\ l^p_{att}) \circ_{nbr(\sigma_{close} y^p_{from},\ l^p_{att})} \mathit{Temp},\ \mathit{if}\ nbr(\sigma_{close} y^p_{from},\ l^p_{att}) \in \mathcal{N}_{\sigma_{close} y^p_{from}}.$$

*If so then the resulted fact* $P$ *is added to the knowledge base of the attacker:* $\mathcal{K} := \mathcal{K} \cup \{P\}$. *Finally, the function* $Attacker_{rep}(\mathcal{K}, \mathcal{A})$ *is called, which represents the behaviour of the attacker node after intercepting the reply.*

7. *Function* $\boldsymbol{resolution}_7(badReq, C^1_4, H)$ *describes the scenario in which the attacker node forwards "its" request to intermediate nodes.*

   $\boldsymbol{resolution}_8(badReq, C^3_4, H)$ *concerns the case when destination node is a neighbor of the attacker node, thus, the attacker node forwards "its" request to the destination.*

   *Function* $\boldsymbol{resolution}_9(badRep, C^2_4)$ *concerns the case in which the attacker node forwards "its" reply to intermediate nodes.*

   *Finally, function* $\boldsymbol{resolution}_{10}(badRep, C^4_4)$ *describes the scenario in which the source node is a neighbor of the attacker node and the attacker forwards "its" reply to the source.*

### 9.5.1 Derivability and derivation diagram

In order to give the definition of derivability and graphical representation of the reasoning in the previous subsection we introduce the notion of *derivation tree.*

**Definition 13.** *We say that* $F$ *is a closed fact if it does not contain any variable.*

For example *accept(honest)*, *accept(advr)* and $route(l^p_{src}, l^p_{dest})$ are closed facts. However, $route(x^p_{src}, x^p_{dest})$ is not closed.

**Definition 14.** *Let* $F$ *be a closed fact. Let* $\mathcal{C}$ *be a set of clauses. A derivation tree of* $F$ *from* $\mathcal{C}$ *is a finite tree defined as follows:*

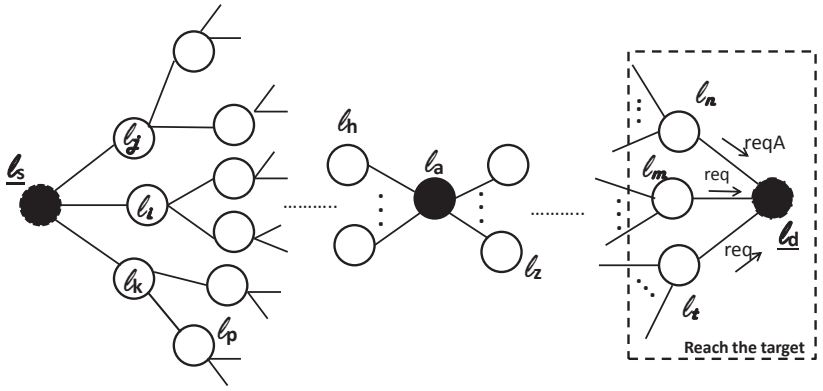


Figure 13: The request reached the destination node.

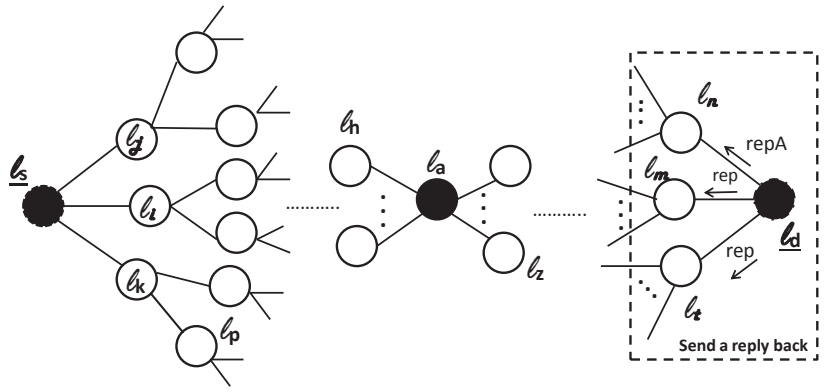


Figure 14: The reply message is generated and sent back by the destination node.

1. *Its nodes are labelled by the name of the rule that is applied in the resolution. In our case these rules can be the protocol rules* $R_1^{req}, \ldots, R_3^{req}$, $R_1^{att}$, $R_2^{att}$, *or the attacker rules* $I_1$, $I_2$, $I_3$, $C_1$, $C_2$, $C_3$, $C_4^1$, $C_4^2$, $C_4^3$, *and* $C_4^4$. *In addition, nodes can be captioned by the pair* $(p, i)$.

2. *Its edges are labelled by the facts using in the resolution steps. Incomming edges represent the hypotheses of the rule that is applied in the resolution steps while the outgoing edge represents the conclusion of the rule.*

   *If the tree contains a node such that:*

   *(i) it is labelled by the rule* $R$, *where* $R = H'_1 \wedge \ldots \wedge H'_n \xrightarrow{p,i} C'$, $H'_i$ $(1 \leq i \leq n)$ *and* $C'$ *are not closed facts; (ii) it has the incoming edges labelled by* $H_1$, $H_2, \ldots$, $H_n$ *and an outgoing edge* $C$, *where* $H_i$ $(1 \leq i \leq n)$

*and $C$ are closed facts.*

*Then the $n$ resolutions $H_i \circ_{H'_i} R$ of the $n$ facts $H_i$ $(1 \leq i \leq n)$ and the rule $R$ are successful with the unifiers $\sigma_i$ $(1 \leq i \leq n)$, and the result of the resolutions is the conclusion $C$, where $C = C'\sigma_1 \ldots \sigma_n$.*

*In short, this means that the closed fact $C$ can be derived from $H_1$, $H_2$,..., $H_n$ using rule $R$.*

The right side of the Figure 15 shows the resolution $route(l_s^p, l_d^p) \circ_{route(x_s^p, x_d^p)} R$, where $R$ is the rule $route(x_s^p, x_d^p) \xrightarrow{p,i} wm(x_s^p, x_{msgreq}^p)$.

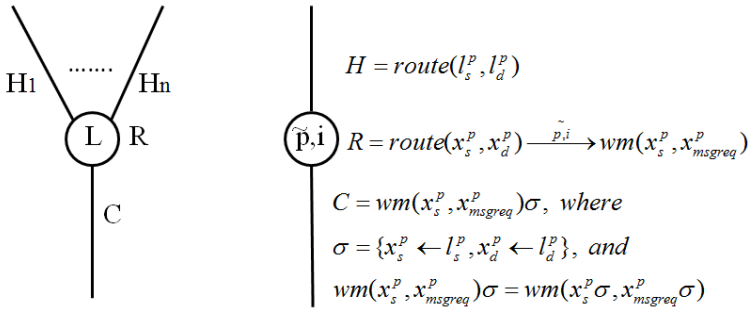


Figure 15: On the left: The derivation of the closed fact $C$ from $H_1$, $H_2$,..., $H_n$ by using rule $R$, where $R = H'_1 \wedge \ldots \wedge H'_n \xrightarrow{p,i} C'$, $H'_i$ $(1 \leq i \leq n)$ and $C'$ are not closed facts On the right: The derivation of the fact $wm(x_s^p, x_{msgreq}^p)\sigma$, where $\sigma = \{x_s^p \leftarrow l_s^p, x_d^p \leftarrow l_d^p\}$, from $route(l_s^p, l_d^p)$ using rule $route(x_s^p, x_d^p) \xrightarrow{p,i} wm(x_s^p, x_{msgreq}^p)$.

### 9.5.2 Reasoning about the attacker activity

The attacker node receives a route request message when the rule $R_1^{att}$ is used in the resolution step, and the resolution step succeeds. For instance we consider the scenario in the Figure 16. In this example, the attacker node $l_a^p$ receives the request from $l_h^p$:
In this case the fact $wm(req)$ resolvable with rule $R_1^{att}$, where

$$R_1^{att} = wm(y_{req}) \wedge nbr(y_{from}^p, l_a^p) \wedge \xrightarrow{p,i_6} att(y_{req}).$$

The resolution yields the rule $Temp_{att}$, $Temp_{att} = nbr(l_h^p, l_a^p) \xrightarrow{p,i_6} att(req)$, where the unifier $\sigma$ is $\{y_{from}^p \leftarrow l_h^p, y_{req} \leftarrow req\}$. Thereafter, the algorithm searches for the fact $nbr(l_h^p, l_a^p)$ in the set $\mathcal{N}_{l_a^p}$. If the fact is found then resolution $nbr(l_h^p, l_a^p) \circ_{nbr(l_h^p, l_a^p)} Temp_{att}$. The fact $att(req)$ is derived as the result of this resolution, which intuitively means that the attacker node $l_a^p$ intercepted the message $req$.

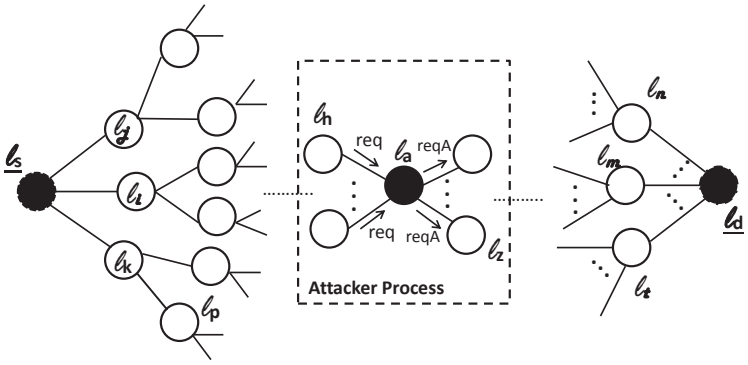


Figure 16: A scenario in which the attacker node receives the route request message *req* from node $l_h$. The request *reqA* is broadcasted by the attacker node.

After receiving the message *req* the attacker can use its initial knowledge (rules $I_1$, $I_2$, $I_3$) and computational ability (rules $C_1$, $C_2$, $C_3$, $C_4^1$, $C_4^2$, $C_4^3$, $C_4^4$ ) to construct messages including an incorrect route.

We restrict the attacker ability to computes at most $n$ new data to use in constructing messages. The value of $n$ varies depend on the system parameter and specified protocols and scenarios.

We note that by restricting the attacker node to construct messages that match the template messages (otherwise, the sent message will not be accepted due to it is not resolvable with template rules $R_i$.) from its knowledge we reduced a large amount of message space the attacker can construct.

**Algorithm 2.** *The functions $Attacker_{req}(\mathcal{K}, \mathcal{A}, H)$ and $Attacker_{rep}(\mathcal{K}, \mathcal{A})$: In this part we give the algorithm describes the behavior of the attacker node. Again, we note that in this paper we consider the case of Active-1-1 attacker only, which means that there is only one active attacker node in the network.*

*The attacker node maintains the set $\mathcal{K}$ of its knowledge. This set includes closed facts of the form $att(p_1, \ldots, p_n)$, and the attacker accumulates the messages it intercepted in $\mathcal{K}$. In addition, it maintains the set $\mathcal{A}$ of attacker rules that represent its computational ability. At the beginning, $\mathcal{K}$ consists of the initial knowledge of the attacker, that is, $\mathcal{K} = \{I_1, I_2, I_3\}$, and $\mathcal{A} = \{ C_1, C_2, C_3, C_4^1, C_4^2, C_4^3, C_4^4 \}$. $\mathcal{K}$ can be updated during the algorithm while $\mathcal{A}$ is fix. We note that this discussion considers the general case, and a specified sets of clauses depeding on the specified routing protocol.*

*When a new $att(p_1, \ldots, p_n)$ closed fact is derived during the route discovery phase it is added to set $\mathcal{K}$. The clause $att(p_1, \ldots, p_n)$ is the result of the resolution step between closed fact $wm(p_1, \ldots, p_m)$ and one of the rules $R_i^{att}$ is successful. Intuitively, this means that the attacker node receives a message broadcasted by a honest node: $wm(p_1, \ldots, p_m)$.*

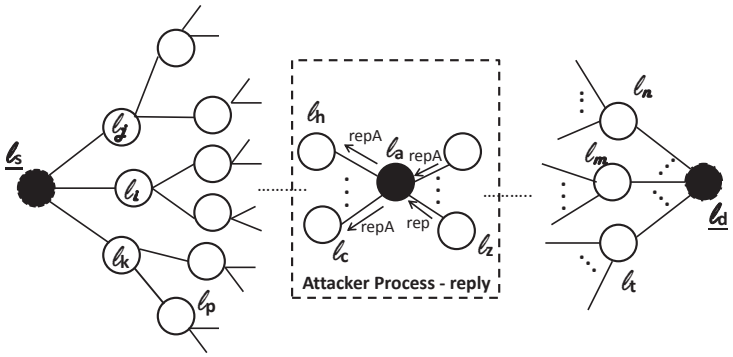


Figure 17: A scenario in which the attacker node receives the route reply message.

*Then using this new information, combining with the other information in* $\mathcal{K}$ *and using computational ability, that is, rules in* $\mathcal{A}$ *it tries to generate a message* $(p_i, \ldots, p_j)$ *that is not equal to the correct message, which honest node should create in his place. Formally this means, closed fact* $att((p_i, \ldots, p_j))$ *is derived from* $\mathcal{K} \cup \mathcal{A}$*, and*

$$att((p_i, \ldots, p_j)) \wedge \neg unifiable\big((p_i, \ldots, p_j), (p'_i, \ldots, p'_j)\big) \longrightarrow \textbf{\textit{badreq}} \quad (Req)$$

*or*

$$att((p_i, \ldots, p_j)) \wedge \neg unifiable\big((p_i, \ldots, p_j), (p'_i, \ldots, p'_j)\big) \longrightarrow \textbf{\textit{badrep}} \quad (Rep)$$

*where closed pattern* $(p'_i, \ldots, p'_j)$ *is a correct message that honest node should generates. The message* $(p'_i, \ldots, p'_j)$ *is depends on what is a specified protocol, and wich kind of attack we want to examine. For instance, in case of the SRP protocol when the attacker node receive message (req, src, dest, ID, MAC,* $l_1, \ldots, l_i$*) then the honest node should insert it* $ID$ *to the end of the message, so that the correct message is (req, src, dest, ID, MAC,* $l_1, \ldots, l_i, l_{att}$*). An other example is when we do not want to examine the relay attack, in which the attacker just re-send the received message unchange, this case the correct message is the message he received. The procedure of computing incorrect request and reply messages are specified by the two functions* $computeBadReq(\mathcal{K}, \{C_1, C_2, C_3\})$ *and* $computeBadRep(\mathcal{K}, \{C_1, C_2, C_3\})$*, respectively. In these two functions if the facts* ***badreq*** *and* ***badrep*** *are derived then the facts* ***badReq*** *and* ***badRep*** *of the form* $att((p_i, \ldots, p_j))$ *are returned, respectively.*

*After creating each such message successfully, the attacker node applies one of the four rules* $C_4^1$*,* $C_4^2$*,* $C_4^3$*,* $C_4^4$ *to forward the message to honest nodes. We emphasize that the attacker tries to construct all possible incorrect message using it accummulated knowledge and computation ability and broadcasts each incorrect message. By properly restricting the ability of the attacker we can examine each possibility within a finite time. We illustrate this by detecting the attack against the SRP protocol in the Section 10.1.*

*We note that in case of specified protocols such as SRP, Adriadne, or EndairA and when we allow only one session of route request (no new interleaving route request is allowed), for reducing a state space the attacker node is restricted to modify only the mutable part of the message such as ID list, digital signature, MAC. The first part of the message usually includes (rep|req, src, dest, ID) tuple, which should not corrupt otherwise it will be dropped by the sorce.*

Now we define the "bad" state that the attacker should not achieve:

**Theorem 2.** *The route discovery from node $l^p_{src}$ to node $l^p_{dest}$ in a given topology $\mathcal{N}$ is corrupted by the attacker node that behaves as in Algorithm 2 if the facts* ***accept(advr)*** *and* ***badreq*** *or* ***badrep*** *(could be both) are derived from the set of clauses $\{route(l^p_{src}, l^p_{dest})$, $R^{req}_1$, $R^{rep}_1$, $R^{req}_2$, $R^{rep}_2$, $R^{req}_3$, $R^{att}_1$, $R^{att}_2$, $R^{att}_3$, $\{\forall l^p_i$: $\mathcal{N}_{l^p_i}\}$, $\mathcal{N}_{l^p_{att}}$ $\} \cup \mathcal{K} \cup \mathcal{A}$.*

*Proof.* First we lay down that in this work we concern such attacker node activity in which the attacker node idle and is activated only when it intercepts some message during the route discovery phase. We do not concern the case when the attacker node initiates the route discovery or when it is the destination node. In addition we do not concern the case when the attacker node itself generates messages and broadcast it.

Three cases can happen for a given tuple ($l^p_s$, $l^p_d$, $l^p_{att}$, $\mathcal{N}$): In the first case there is no route from $l^p_s$ to $l^p_d$ that includes the attacker node $l^p_{att}$. This case is illustrated in the Figure 18.

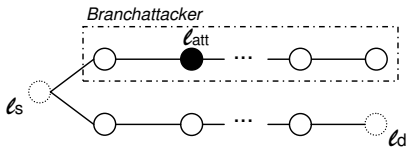


Figure 18: *The scenario in which the attacker node is not in the route between the source and the destination nodes.*

Let us recall that the deductive algorithm restricts that only such the reply messages can reach the source node $l^p_{src}$ which are sent by the attacker node $l^p_{att}$. So that the fact **accept(advr)** is only derivable when the source node accepts the reply sent by the attacker node. However, the algorithm allows request messages sent by honest nodes propagate from the source to the destination node (may be through the attacker node) freely, because we want to model the attacker that collects several request messages it intercepts and computing the message containing an incorrect route based on the collected information. The reply messages forwarded by honest nodes are allowed to propagates until they reach the attacker node. After that the algorithm considers only the reply messages sent the attacker node.

According to the deductive algorithm. In the first phase, the request message propagates from the source node to the attacker node. Then in the second

phase the function *Attacker* is called. Let us recall that the function *Attacker* is called once a "new" request/reply message is intercepted by the attacker node. In this scenario, the attacker node $l^p_{att}$ can construct the message containing an incorrect route, that is, the facts ***badreq*** or ***badrep*** (could be both) is derivable. The fact **accept(advr)** is derived only when the attacker node ables to construct itself the reply message containing an incorrect route that is then accepted by the algorithm.

In the second case the attacker node is in the route between the source node $l^p_{src}$ and the destination node $l^p_{dest}$. This scenario is illustrated the Figure 19.

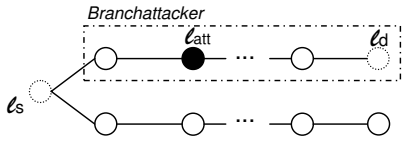


Figure 19: *The scenario in which the attacker node is in the route between the source and the destination nodes.*

In this scenario beside receiving the request the attacker node also receives the reply message sent by honest nodes. So the bad state happens when the attacker can construct incorrect message(s) which is then accepted by $l^p_{src}$, that is, the facts **accept(advr)** and ***badreq*** or ***badrep*** are derived.

Finally, in the third case there is no any route from the node $l^p_{src}$ to the attacker node $l^p_{att}$. In this scenario the attacker node does not intercepts any message, hence, the function *Attacker* is not called and the facts ***badreq*** and ***badrep*** are not derived. The route discovery is then executed among honest nodes only, which means that no attack is performed in this case.

□

Next we give the definition of security of routing protocols: The first definition concern the security for a given network topology.

**Definition 15.** *We say that a routing protocol is secure for a given topology $\mathcal{N}$ within n computation steps (i.e., n resolution steps) of the attacker node if for all possible tuples ($l^p_{src}$, $l^p_{dest}$, $l^p_{att}$) if the route discovery from node $l^p_{src}$ to node $l^p_{dest}$ in $\mathcal{N}$ is **not** corrupted by the attacker node that behaves as in Algorithm 2.*

and the stronger definition for all possible topology:

**Definition 16.** *We say that a routing protocol is secure within n computation steps of the attacker node (i.e., n resolution steps made by the algorithm) if for all possible tuples ($\mathcal{N}$, $l^p_{src}$, $l^p_{dest}$, $l^p_{att}$) if the route discovery from node $l^p_{src}$ to node $l^p_{dest}$ in $\mathcal{N}$ is **not** corrupted by the attacker node that behaves as in Algorithm 2.*

**Theorem 3.** *Let $\mathcal{B}$ be the finite computation steps (resolution steps) the attacker $\mathcal{M}_A$ can made. If $\mathcal{M}_A$ can perform attacks against the routing protocol*

*for a given network topology $\mathcal{N}$ and tuple ($l^p_{src}$, $l^p_{dest}$, $l^p_{att}$) then the algorithm will find an attack.*

*Proof.* Again let us recall that in this work we concern such attacker node activity in which the attacker node idle and is activated only when it intercepts some message during the route discovery phase. We do not concern the case when the attacker node initiates the route discovery or when it is the destination node. In addition we do not concern the case when the attacker node itself generates messages and broadcast it.

The tuple ($l^p_{src}$, $l^p_{dest}$, $l^p_{att}$) in our statement represents the identifier of the source, the destination, the attacker node, and the ability of the attacker (knowledge, and computation ability). That is, the attack is found for a given attacker ability.

In addition, let us recall that the attacker tries to construct all (exhaustively) possible incorrect message using its accummulated knowledge and computation ability and then broadcasts each incorrect message.

Let us "indirectly" suppose that there is an attack in the scenario ($l^p_{src}$, $l^p_{dest}$, $l_{att}$, $\mathcal{N}$) that the algorithm does not find. We will show that this leads to a contradiction.

For a given scenario ($l^p_{src}$, $l^p_{dest}$, $l_{att}$, $\mathcal{N}$) there are three cases may happen as discussed in the Theorem 2. The attack may be performed only in the first two cases. In these scenarios an attack may happen when incorrect messages are constructed by the attacker node that are then accepted by the source node. However, according to the Theorem 2 the examining of these cases are all performed by the deductive algorithm. □

**Theorem 4.** *Let us assume that $Verif^{facts}_{init}$, $Verif^{facts}_{interm}$ and $Verif^{facts}_{dest}$ are empty. The number of resolution steps made by the algorithm is upper bounded by $(\mathcal{D}+2)(|N|+|E|)+\mathcal{B}$, where $|N|$, $|E|$ are the number of nodes and edges of the topology; $\mathcal{D}$ is the total number of incorrect messages created by the attacker during the algorithm that are accepted by its honest neighbors; and $\mathcal{B}$ is the total number of the resolution steps (computation steps) made by the attacker during the algorithm. For practical reasons both $\mathcal{D}$ and $\mathcal{B}$ are assumed to be finite.*

*Proof.* **(sketch)** The propagation of a request from the source to the destination can be seen as a breadth-first traversing in the graph. In case there is no attacker in the network, the algorithm performs at most $|N|+|E|$ resolution steps. The same is true regarding the propagation of the reply. Note that in most cases the reply is returned by unicast sending instead of broadcast, thus, the number of resolution steps equals to the length of the route. Taking into account the attacker node, whenever it intercepts a new request/reply it attempts to compute incorrect messages that will be accepted by its honest neighbors. These incorrect messages will propagate to the source/destination, which takes at most $\mathcal{D}(|N|+|E|)$ resolution steps. Finally, the total resolution steps made by the attacker to construct incorrect messages is $\mathcal{B}$. □

We note that in case $Verif^{facts}_{init}$, $Verif^{facts}_{interm}$ and $Verif^{facts}_{dest}$ are not empty, the complexity depends on the maximal number and the type of the facts they

contain. In a general case the complexity 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 as in case the SRP protocol.

**Theorem 5.** ***(Termination)*** *The deductive algorithm terminates after finite steps if the computation steps the attacker node performs is finite.*

*Proof.* The deductive algorithm is based on guided resolution steps performing a breadth first search. Due to the number of the nodes in the network is finite the breadth first search is finite. This remains finite if the compuatation steps performed by the attacker node is finite. □

We note that by properly restricting the attacker's computation ability, as in case of the SRP protocol, the algorithm terminates after finite resolution steps.

# 10 Application of our automatic verification method

In this section we demonstrate how to apply our automatic verification technique to detect the attack on the SRP protocol that we have analysed manually above.

## 10.1 Verifying the SRP protocol

As the first step we specify the SRP protocol by the process $P_i$ as follows:

$$P_i \stackrel{def}{=} !ReqInit^{P_i}(l_x) |\ !Interm^{P_i}(l_x)\ |\ !Dest^{P_i}(l_x)$$

The parameter $l_x$ is a variable for specifying the node identifier of the current node of which the operation is described. It is a kind of reference to "this" node. When describing the operation of intermediate nodes we distinguish two cases: (i) a given intermediate node is a neighbor of the source node, this scenario is defined by the process $Listen_1^{P_i}(l_x)$ and (ii) a given intermediate node is *not* a neighbor of the source node, this scenario is defined by the process $Listen_2^{P_i}(l_x)$.

Hence, we define the process $Interm^{P_i}(l_x)$ as $Listen_1^{P_i}(l_x)\ |\ Listen_2^{P_i}(l_x)$. Then, we give the definition of each process as follows:

$$\begin{array}{l} ReqInit^{P_i}(l_x) \stackrel{def}{=} \\ \quad c_{p_i}(x_{dst}).\nu id\ (let\ MAC_1 = mac((req, l_x, x_{dst}, id), k(l_x, x_{dst})) \\ \quad in\ \langle (req, l_x, x_{dst}, id, MAC_1)\rangle.!WaitRep_{reqinit}^{P_i}). \end{array}$$

This process is invoked when the node $l_x$ is the source node which initiates the route discovery towards the destination. Formally, the node receives the identifier of the destination and creates the new message identifier *id*. Afterwards it creates the message authentication code and includes it into the request message. Finally it broadcasts the request message and waits for the reply. The process $WaitRep_{reqinit}^{P_i}$ is defined as:

$$WaitRep_{reqinit}^{P_i} \stackrel{def}{=}$$

$(= l_x, (= rep, = l_x, = x_{dst}, = id, = [x_{next}, List],$
$= mac((rep, l_x, x_{dst}, id, [x_{next}, List])), k(l_x, x_{dst}))).\langle ACCEPT\rangle$

The SRP protocol is described in the syntax of processes in the *sr*-calculus. In order to shorten the length of the protocol description we add some syntactic sugar similar to the syntactic sugar used in the case of the Proverif tool. In particular, instead of writing for example $(x).[x = t]P$ we write $(= t).P$. Intuitively, $(= t).P$ means that if the input is equal to $t$ then the process $P$ is executed otherwise it stucks. Hence, this process functionates as follows: If some message is received verification steps are made and of all verificaion steps pass then the special term $ACCEPT$ is signalled. The process $Listen_1^{P_i}(l_x)$ is invoked when the node $l_x$ is an intermediate node, and is defined as follows:

$Listen_1^{P_i}(l_x) \stackrel{def}{=}$
$((= req, y_{src}, y_{src}, y_{id}, y_{mac})).$
$\langle (req, y_{src}, y_{dst}, y_{id}, y_{mac}, [l_x])\rangle.!\, WaitRep_{listen}^{P_i}.$

Intuitively, this means that if some message is received by the node then it checks whether it is the request meassage. If so then it appends its identifier to the message it received and broadcasts it and then waits for the reply message:

$WaitRep_{listen}^{P_i} \stackrel{def}{=}$
$(= l_x, (= rep, = y_{src}, = y_{dst}, = y_{id}, = [l_x, y_{next}, List_2], y'_{mac})).$
$\langle y_{src}, (rep, y_{src}, y_{dst}, y_{id}, [l_x, y_{next}, List_2], y'_{mac})\rangle.$

The operation of process is similar to the process $WaitRep_{reqinit}^{P_i}$ excepts that intermediate nodes do not verify the message authentication code. This is represented by the variable $y'_{mac}$, which can be unified with any terms (which could be fake message authentication code), that is, the node will accepts it whatever it is. Further, the $WaitRep_{reqinit}^{P_i}$ represents the scenario in which the intermediate node is the neighbor of the source node. That is, the it waits for the list of identifier of the form $[l_x, y_{next}, List_2]$ in which there is node identifier before $l_x$.

The process $Listen_2^{P_i}$ is modelled the another possible operation of an intermediate node:

$Listen_2^{P_i}(l_x) \stackrel{def}{=}$
$((= req, y_{src}, y_{dst}, y_{id}, y_{mac}, [List, y_{prev}])).$
$\langle (req, y_{src}, y_{dst}, y_{id}, y_{mac}, [List, y_{prev}, l_x])\rangle.!\, WaitRep_{listen}^{P_i}.$

This process differs from the process $Listen_1^{P_i}(l_x)$ in that it models the scenario in which a node receives the request message in which the list of identifiers contains at least one identifier, while in case of the process $Listen_1^{P_i}(l_x)$ the request message including the empty list of identifier is received.

The process $WaitRep_{listen}^{P_i}$ is defined as $(WaitRep_1^{FW} \,|\, WaitRep_2^{FW})$. $WaitRep_1^{FW}$ corresponds to the scenario in which the intermediate node $l_x$ is not a neighbor

of the source and the destination node, that is, in the returned reply message the list of the identifier is $[List_1, y_{prev}, l_x, y_{next}, List_2]$, where $y_{prev}$ and $y_{next}$ are the identifier of other intermediate nodes.

$$
\begin{array}{l}
WaitRep_1^{FW} \stackrel{def}{=} \\
\quad (= l_x, (= rep, = y_{src}, = y_{dst}, = y_{id}, = [List_1, y_{prev}, l_x, y_{next}, List_2], y'_{mac})). \\
\quad \langle y_{prev}, (rep, y_{src}, y_{dst}, y_{id}, [List_1, y_{prev}, l_x, y_{next}, List_2], y'_{mac})\rangle.
\end{array}
$$

The process $WaitRep_2^{FW}$ concerns the scenario in which the node $l_x$ is the neighbor of the destination node, which means that there is no node indentifier after $l_x$ in the list of identifier.

$$
\begin{array}{l}
WaitRep_2^{FW} \stackrel{def}{=} \\
\quad (= l_x, (= rep, = y_{src}, = y_{dst}, = y_{id}, = [List, y_{prev}, l_x], y'_{mac})). \\
\quad \langle y_{prev}, (rep, y_{src}, y_{dst}, y_{id}, [List, y_{prev}, l_x], y'_{mac})\rangle.
\end{array}
$$

Finally, the process $Dest^{P_i}(l_x)$ model the operation of the destination node. When receives some message the destination node checks if the message is the request, the message authentication code is correct. If so then is generates a reply message and sends back to the node of which identifier is the last element of the list of identifier.

$$
\begin{array}{l}
Dest^{P_i}(l_x) \stackrel{def}{=} \\
\quad ((= req, y_{src}, = l_x, y_{id}, = mac((req, y_{src}, l_x, y_{id}, k(y_{src}, l_x)), [List, y_{prev}]))\,. \\
\quad let\ MAC_1 = mac\,((rep, y_{src}, l_i, y_{id}, [List, y_{prev}]), k(y_{src}, l_x))\ \ in \\
\quad \langle (rep, y_{src}, l_x, y_{id}, [List, y_{prev}], MAC_1)\rangle.
\end{array}
$$

This description is then translated to the following protocol rules described in clauses:

$$
\frac{R_1^{req}}{route(x^p_{l_x}, x^p_{dest}) \longrightarrow wm\left(\left(req^p, x^p_{l_x}, x^p_{dest}, ID[], mac\left((req^p, x^p_{l_x}, x^p_{dest}, ID[]), k(x^p_{l_x}, x^p_{dest})\right)\right), honest\right)}
$$

The rule $R_1^{req}$ models the case when the source node $x^p_{l_x}$ creates a request message $\left((req^p, x^p_{l_x}, x^p_{dest}, x^p_{ID}, mac\left((req^p, x^p_{l_x}, x^p_{dest}, x^p_{ID}), k(x^p_{l_x}, x^p_{dest})\right)\right)\right)$.

$$
\begin{array}{l}
\frac{R_1^{rep}}{wm\left(\left(x^p_{l_x}, rep^p, x^p_{l_x}, x^p_{dest}, x^p_{ID}, [x^p_{next}, List^p], mac\left((rep^p, x^p_{src}, x^p_{l_x}, x^p_{ID}, [x^p_{next}, List^p]), k(x^p_{l_x}, x^p_{dest})\right)\right), s\right)} \\
\wedge\ nbr(x^p_{l_x}, x^p_{next}) \stackrel{p,i_2}{\longrightarrow} \textbf{\textit{accept(s)}}
\end{array}
$$

The rule $R_1^{rep}$ models the case when the source node $x^p_{l_x}$ receives some message. If this message is unifiable with the message $\left(x^p_{l_x}, rep^p, x^p_{l_x}, x^p_{dest}, x^p_{ID}, [x^p_{next}, List^p], mac\left((rep^p, x^p_{src}, x^p_{l_x}, x^p_{ID}, [x^p_{next}, List^p]), k(x^p_{l_x}, x^p_{dest})\right)\right)$ then the fact ***ok(s)*** is derived, which means that the source node accepted the returned route. If $s$=*advr* then it means that the source node accepted the incorrect route while $s$=*honest* means that the reply message is not forwarded

by the attacker node but only honest nodes.

$$\underline{R_2^{req^1}}$$

$$wm\left(\left(req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}\right), s\right) \wedge nbr(y^p_{src}, y^p_{l_x}) \xrightarrow{p, i^1_3} wm\left(\left(req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [y^p_{l_x}]\right), s\right)$$

The rule $R_2^{req^1}$ says that if the route request message is broadcasted by the source node then the neighbors of the source node receive the request and append their identifier to the request and re-broadcast the modified request.

$$\underline{R_1^{att^1}}$$

$$wm\left(\left(req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}\right), s\right) \wedge nbr(y^p_{src}, l^p_{att}) \wedge \xrightarrow{p, i^1_6} att\left(\left(req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}\right), s\right)$$

The rule $R_1^{att^1}$ says that if the route request message is broadcasted by the source node and the attacker node is the neighbor of the source node then the attacker intercepts the request.

$$\underline{R_2^{rep^1}}$$

$$wm\left(\left(y^p_{l_x}, rep^p, y^p_{src}, y^p_{dest}, y^p_{ID}, [y^p_{l_x}, y^p_{next}, List^p], y'^p_{mac}\right), s\right) \wedge nbr(y^p_{l_x}, y^p_{next}) \xrightarrow{p, i^1_4}$$
$$wm\left(\left(y^p_{src}, rep^p, y^p_{src}, y^p_{dest}, y^p_{ID}, [y^p_{l_x}, y^p_{next}, List^p], y'^p_{mac}\right), s\right)$$

The rule $R_2^{rep^1}$ models scenario in which the honest intermediate node $y^p_{l_x}$ receives a reply message. This rule says that if the message the node $y^p_{l_x}$ received is unifiable with the message $\left(y^p_{l_x}, rep^p, y^p_{src}, y^p_{dst}, y^p_{ID}, [y^p_{l_x}, y^p_{next}, List^p], y'^p_{mac}\right)$, and the node $y^p_{next}$ is the neighbor of the node $y^p_{l_x}$ then node $y^p_{l_x}$ forwards the reply message to the source node $y^p_{src}$.

$$\underline{R_2^{req^2}}$$

$$wm\left(\left(req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [List^p, y^p_{prev}]\right), s\right) \wedge nbr(y^p_{prev}, y^p_{l_i}) \xrightarrow{\tilde{p}, i^2_3}$$
$$wm\left(\left(req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [List^p, y^p_{prev}, y^p_{l_i}]\right), s\right)$$

The rule $R_2^{req^2}$ says that if the route request message is broadcasted by an intermediate node then its neighbors receive the request and append their identifier to the request and re-broadcast the modified request.

$$\underline{R_1^{att^2}}$$

$$wm\left(\left(req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [List^p, y^p_{prev}]\right)\right) \wedge nbr(y^p_{prev}, l^p_{att}) \xrightarrow{p, i^2_6} att\left(\left(req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [List^p, y^p_{prev}]\right)\right)$$

The rule $R_1^{att^2}$ says that if the route request message is broadcasted by an intermediate node and the attacker node is its neighbor then the attacker intercepts the request.

$$\underline{R_2^{rep^2}}$$

$$wm\left(\left(y^p_{l_x}, rep^p, y^p_{src}, y^p_{dest}, y^p_{ID}, [List^p_1, y^p_{prev}, y^p_{l_x}, y^p_{next}, List^p_2], y'^p_{mac}\right), s\right) \wedge nbr(y^p_{l_x}, y^p_{next}) \wedge$$
$$nbr(y^p_{l_x}, y^p_{prev}) \wedge \xrightarrow{p, i^2_4} wm\left(\left(y^p_{prev}, rep^p, y^p_{src}, y^p_{dest}, y^p_{ID}, [List^p_1, y^p_{prev}, y^p_{l_x}, y^p_{next}, List^p_2], y'^p_{mac}\right), s\right)$$

The rule $R_2^{rep^2}$ models scenario in which the honest intermediate node $y_{l_x}^p$ receives a reply message. This rule says that if the message the node $y_{l_x}^p$ received is unifiable with the message $\left(y_{l_x}^p, rep^p, y_{src}^p, y_{dest}^p, y_{ID}^p, [List_1^p, y_{prev}^p, y_{l_x}^p, y_{next}^p, List_2^p], y_{mac}'^p\right)$, and the nodes $y_{prev}^p$ and $y_{next}^p$ are the neighbors of the node $y_{l_x}^p$ then node $y_{l_x}^p$ forwards the reply message to node $y_{prev}^p$.

$$\frac{R_3^{req}}{wm\left(\left(req^p, z_{src}^p, x_{dest}^p, z_{ID}^p, mac((rep^p, z_{src}^p, x_{dest}^p, z_{ID}^p, k(z_{src}^p, x_{dest}^p)), [List^p, z_{prev}^p]\right), s\right) \wedge nbr(z_{prev}^p, x_{dest}^p)}$$
$$\xrightarrow{p,i_5} wm\left(\left(z_{prev}^p, rep^p, z_{src}^p, x_{dest}^p, z_{ID}^p, [List^p, z_{prev}^p], mac\left((rep^p, z_{src}^p, x_{dest}^p, z_{ID}^p, [List^p, z_{prev}^p]), k(z_{src}^p, x_{dest}^p)\right)\right), s\right)$$

The rule $R_3^{req}$ says that if the route request message is broadcasted by a honest intermediate node and the destination node $x_{dest}^p$ is its neighbor then the destination node receives the request and sends back the reply message.

$$\frac{R_2^{rep^3}}{wm\left(\left(y_{l_x}^p, rep^p, y_{src}^p, y_{dst}^p, y_{ID}^p, [List_1^p, y_{prev}^p, y_{l_x}^p], y_{mac}'^p\right), s\right) \wedge nbr(y_{l_x}^p, y_{prev}^p) \xrightarrow{\bar{p},i_4^3}}$$
$$wm\left(\left(y_{prev}^p, (rep^p, y_{src}^p, y_{dst}^p, y_{ID}^p, [List_1^p, y_{prev}^p, y_{l_x}^p], y_{mac}'^p)\right), s\right)$$

The rule $R_2^{rep^3}$ says that when the reply message is received by a neighbor of the destination node. After verifying that the node $y_{prev}^p$ is its neighbor the reply message is forwarded to the node $y_{prev}^p$.

$$\frac{R_2^{att}}{wm\left(\left(y_{l_x}^p, rep^p, y_{src}^p, y_{dst}^p, y_{ID}^p, [List_1^p, y_{prev}^p, y_{l_x}^p, y_{next}^p, List_2^p], y_{mac}'^p\right), s\right) \wedge nbr(y_{l_x}^p, y_{next}^p) \wedge}$$
$$nbr(y_{l_x}^p, y_{prev}^p) \wedge nbr(y_{l_x}^p, l_{att}^p) \xrightarrow{p,i_7} att\left(\left(y_{prev}^p, rep^p, y_{src}^p, y_{dst}^p, y_{ID}^p, [List_1^p, y_{prev}^p, y_{l_x}^p, y_{next}^p, List_2^p], y_{mac}'^p\right), s\right)$$

The rule $R_2^{att}$ says that if an intermediate node $y_{l_x}^p$ forwards the reply message to the node $y_{prev}^p$ and the attacker node $l_{att}^p$ is its neighbor then the attacker node intercepts the reply.

The node identifiers of the five nodes are the constants $l_1^p$, $l_2^p$, $l_3^p$, $l_4^p$, and $l_{att}^p$. The input of the algorithm is the tuple ($\mathcal{T}_0$, $\{\mathcal{N}_{l_i^p} | 1 \leq i \leq 4\}$, $\mathcal{N}_{l_{att}^p}$, $route(l_1^p, l_3^p)$, $\mathcal{K}$, $\mathcal{A}$) where: $\mathcal{T}_0$ contains 11 rules in total including the rules $R_1^{req}$, $R_2^{req^1}$ and $R_1^{att^2}$ that we use in this demonstration. The topology is defined by the five sets $\mathcal{N}_{l_1^p}$, $\mathcal{N}_{l_2^p}$, $\mathcal{N}_{l_3^p}$, $\mathcal{N}_{l_4^p}$, and $\mathcal{N}_{l_{att}^p}$ where : $\mathcal{N}_{l_1^p} = \{nbr(l_1^p, l_2^p),\ nbr(l_1^p, l_{att}^p)\}$; $\mathcal{N}_{l_2^p} = \{nbr(l_2^p, l_1^p),\ nbr(l_2^p, l_{att}^p)\}$; $\mathcal{N}_{l_3^p} = \{nbr(l_3^p, l_{att}^p),\ nbr(l_3^p, l_4^p)\}$; $\mathcal{N}_{l_4^p} = \{nbr(l_4^p, l_{att}^p),\ nbr(l_4^p, l_3^p)\}$; $\mathcal{N}_{l_{att}^p} = \{nbr(l_{att}^p, l_1^p),\ nbr(l_{att}^p, l_2^p),\ nbr(l_{att}^p, l_3^p),\ nbr(l_{att}^p, l_4^p)\}$. The initial knowledge $\mathcal{K}$ and the computational ability $\mathcal{A}$ of the attacker are defined by the two sets $\{I_1,\ I_2,\ I_3\}$ and $\{S_1, S_2, S_3, S_4, S_5\}$, respectively. The initial knowledge is the same as in Section 9.4. The rules $S_1$, $S_2$, $S_3$, $S_4$ and $S_5$ are specified as follows:

$S_1.\ att(i) \rightarrow att(n[i])$

Rule $S_1$ is the same as rule $C_1$ in Section 9.4.

$$S_2.\ att\left((req^p, x^p_{src}, x^p_{dest}, x^p_{ID}, x^p_{mac}, List^p)\right) \wedge att(y^p_l) \rightarrow att((req^p, x^p_{src}, x^p_{dest}, x^p_{ID}, x^p_{mac}, [List^p, y^p_l]))$$

Rule $S_2$ is the special case of rule $C_2$ and says that the attacker node can append any data it has to the end of the ID list embedded in the request it receives. Pattern $List^p$ represents an ID list, which can be empty. The variables $x^p_{src}$, $x^p_{dest}$, $x^p_{ID}$, and $x^p_{mac}$ specify the ID of the source and destination node, the message ID, and the message authentication code, respectively.

$$S_3.\ att((x^p_l, rep^p, x^p_{src}, x^p_{dest}, x^p_{ID}, List^p, x^p_{mac})) \wedge att(y^p_l) \rightarrow att((y^p_l, rep^p, x^p_{src}, x^p_{dest}, x^p_{ID}, List^p, x^p_{mac}))$$

Rule $S_3$ is an another special case of rule $C_2$ and says that if the attacker receives a reply message $(rep^p, x^p_{src}, x^p_{dest}, x^p_{ID}, List^p, x^p_{mac})$ addressed to node $x^p_l$ then it can replace the node identifier at the beginning of the message, which specifies the addressee, by an another identifier $y^p_l$. This rule intend to model that the attacker can forward the reply in the name of another nodes.

$$S_4.\ att\left(req^p, x^p_{src}, x^p_{dest}, x^p_{ID}, {MAC}^{req}, [List^p, x^p_{prev}]\right) \wedge nbr(l^p_{att}, x^p_{dest}) \wedge nbr(x^p_{dest}, l^p_{att}) \wedge nbr(x^p_{dest}, x^p_{prev}) \xrightarrow{p,i_{10}} att\left((x^p_{prev}, rep^p, x^p_{src}, x^p_{dest}, x^p_{ID}, [List^p, x^p_{prev}], {MAC_1}^{rep}), honest\right).$$

where ${MAC}^{req}$ is $mac((rep^p, x^p_{src}, x^p_{dest}, x^p_{ID}, k(x^p_{src}, x^p_{dest}))$ and ${MAC_1}^{rep}$ is $mac\left((rep^p, x^p_{src}, x^p_{dest}, x^p_{ID}, [List^p, x^p_{prev}]), k(x^p_{src}, x^p_{dest})\right)$. Rule $S_4$ is the special case of rule $C^3_4$ concerning the case when the destination node and the attacker node are neighbors of each other. In this special case $Verif^{facts}_{att}$ is $nbr(x^p_{dest}, x^p_{prev})$ that models the verification step in which the destination checks if the last ID in the ID list belongs to its neighbor.

$$S_5.\ att((^p_{src}, rep^p, x^p_{src}, x^p_{dest}, x^p_{ID}, [x^p_{next}, List^p], {MAC_2}^{rep}), s) \wedge nbr(l^p_{att}, x^p_{src}) \wedge nbr(x^p_{src}, x^p_{next}) \xrightarrow{p,i_{11}} \mathbf{accept(advr)}.$$

where ${MAC_2}^{rep}$ is $mac\left((rep^p, x^p_{src}, x^p_{dest}, x^p_{ID}, List^p), k(x^p_{src}, x^p_{dest})\right)$. Rule $S_5$ is the special case of rule $C^4_4$ concerning the case when the source node is a neighbor of the attacker node.

The most important protocol rules that we use in this demonstration are the rules $R^{req}_1$, ${R^{req}_2}^1$ and ${R^{att}_1}^2$:

$$R^{req}_1 = route(x^p_{src}, x^p_{dest}) \longrightarrow wm(req^p, x^p_{src}, x^p_{dest}, ID, mac((req^p, x^p_{src}, x^p_{dest}, ID, k(x^p_{src}, x^p_{dest})), [\,])$$

Rule $R^{req}_1$ models the scenario when the source node $x^p_{src}$ creates and broadcasts the request message $(req^p, x^p_{src}, x^p_{dest}, x^p_{ID}, mac\left((req^p, x^p_{src}, x^p_{dest}, x^p_{ID}), k(x^p_{src}, x^p_{dest})\right))$.

$${R^{req}_2}^1 = wm\left((req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [\,])\right) \wedge nbr(y^p_{src}, y^p_{l_x}) \xrightarrow{p,i^1_2} wm\left((req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [y^p_{l_x}])\right)$$

Rule $R_2^{req^1}$ says that if the route request message is broadcasted by the source node then the honest neighbors of the source node receive the request and they append their own identifier to the request and re-broadcast the modified request.

$$R_1^{att^2} = wm\left((req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [List^p, y^p_{prev}])\right) \wedge nbr(y^p_{prev}, l^p_{att}) \xrightarrow{p,i_6} att\left((req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [List^p, y^p_{prev}])\right)$$

Rule $R_1^{att^2}$ says that if a route request message is broadcasted by an intermediate node and the attacker node is its neighbor then the attacker intercepts that request. $List^p$ is a pattern that represent a list of node identifiers, which can be empty.

The derivation of one possible attack against the SRP protocol is modelled by derivation trees shown in the Figure 20 and Figure 21. Figure 20 describes the propagation of the request message from the source node $l_1^p$ to the attacker node $l_{att}^p$ via the intermediate node $l_2^p$. First, the resolution $route(l_1^p, l_3^p) \circ_{route(x^p_{src}, x^p_{dest})} R_1^{req}$ is computed. With the unifier $\sigma_1$, $\sigma_1$=$\{x^p_{src} \leftarrow l_1^p, x^p_{dest} \leftarrow l_3^p\}$ $route(l_1^p, l_3^p) \circ_{route(x^p_{src}, x^p_{dest})} R_1^{req}$ yields the fact

$$wm((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [\,])),$$

which is then resolved with $R_2^{req^1}$ and yields the rule $R_{tmp}^1$:

$$R_{tmp}^1 = nbr(l_1^p, y^p_{l_x}) \xrightarrow{p,i_2} wm\left((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [y^p_{l_x}])\right)$$

where the unifier $\sigma_2$ of the above resolution is

$$\sigma_2 = \{y^p_{src} \leftarrow l_1^p,\ y^p_{dest} \leftarrow l_3^p,\ y^p_{ID} \leftarrow ID,\ y^p_{mac} \leftarrow mac\,((req^p, l_1^p, l_3^p, ID), k(l_1^p, l_3^p))\}.$$

Afterwards, the facts in the set $\mathcal{N}_{l_1^p}$ are resolved with $R_{tmp}^1$. The resolution $nbr(l_1^p, l_2^p) \circ_{nbr(l_1^p, y^p_{l_x})} R_{tmp}^1$ with the unifier $\sigma_3$, $\sigma_3$=$\{y^p_{l_x} \leftarrow l_2^p\}$ yields the fact $wm((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p]))$. Intuitively, this means that node $l_2^p$ received the request message broadcasted by $l_1^p$, and then $l_2^p$ appends its identifier to the request and re-broadcasts it.

The following resolution steps model the case when the attacker node intercepts the request broadcasted by node $l_2^p$: The resolution

$$wm((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p])) \circ_F R_2^{att}$$

where $F$ is $wm\left((req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [List^p, y^p_{prev}])\right)$ and the the unifier $\sigma_4$ is $\sigma_2 \cup \{List^p \leftarrow [\,],\ y^p_{prev} \leftarrow l_2^p\}$. As the result we get the rule $R_{tmp}^2$:

$$R_{tmp}^2 = nbr(l_2^p, l_{att}^p) \xrightarrow{p,i_6} att\,((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p]))$$

Finally, the algorithm searches for the fact $nbr(l_2^p, l_{att}^p)$ in the set $\mathcal{N}_{l_2^p}$. When $nbr(l_2^p, l_{att}^p)$ is found the resolution $nbr(l_2^p, l_{att}^p) \circ_{nbr(l_2^p, l_{att}^p)} R_{tmp}^2$ is computed, which yields the fact $att\,((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p]))$.

Figure 21 describes the behaviour of the attacker node after intercepting the message $(req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p])$: First, the attacker creates a fake identifier $n[i]$ by rule $S_1$. Afterwards, the fake identifier is appended to the ID list $[l_2^p]$, this step is modelled by the two resolutions

$$att\left((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p])\right) \circ_F S_2 \text{ and}$$
$$att(n[i]) \circ_{att(y^p)} R^3_{tmp}$$

where $R^3_{tmp}$ is the result of the first resolution.

Thereafter, the attacker appends the identifier $l_4^p$ to the list $[l_2^p, n[i]]$. This is modelled by the two resolutions

$$att\left((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p, n[i]])\right) \circ_F S_2 \text{ and}$$
$$att(l_4^p) \circ_{att(y^p)} R^4_{tmp}$$

where $R^4_{tmp}$ is the result of the first resolution. We note that $att(l_4^p)$ is an element of the set $I_1$ that is a part of the initial knowledge of the attacker node.

Then the attacker node broadcasts the modified request, which is received by the destination node $l_3^p$. The destination node accepts the message sent by the attacker node and generates the reply message. Afterwards, the destination node sends back the reply to the node $l_4^p$, which is overheared by the attacker node. This step is modelled by the resolutions involving the rule $S_2$. On receiving the reply the attacker replace $l_4^p$ by $l_1^p$. This step is modelled by the resolution steps involving the rule $S_3$. Finally, the attacker node forwards the reply to the source node in the name of the node $l_2^p$. This step is modelled by resolutions using the rule $S_5$.

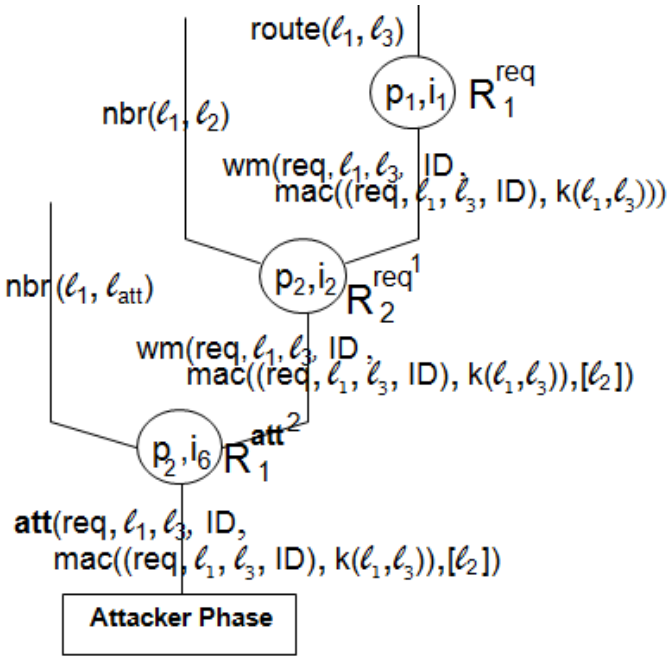


Figure 20: *This part of the derivation tree describes the resolution steps that represent the propagation of the request message from the source node* $l_1^p$ *to the attacker node* $l_{att}^p$.

# 11 Conclusion and future work

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[1].

[1]For the proposed solution for this problem, please see "Ta Vinh Thong, L. Buttyán, On automating the verification of secure ad-hoc network routing protocols, Telecommunication Systems Journal, Springer, 2011"

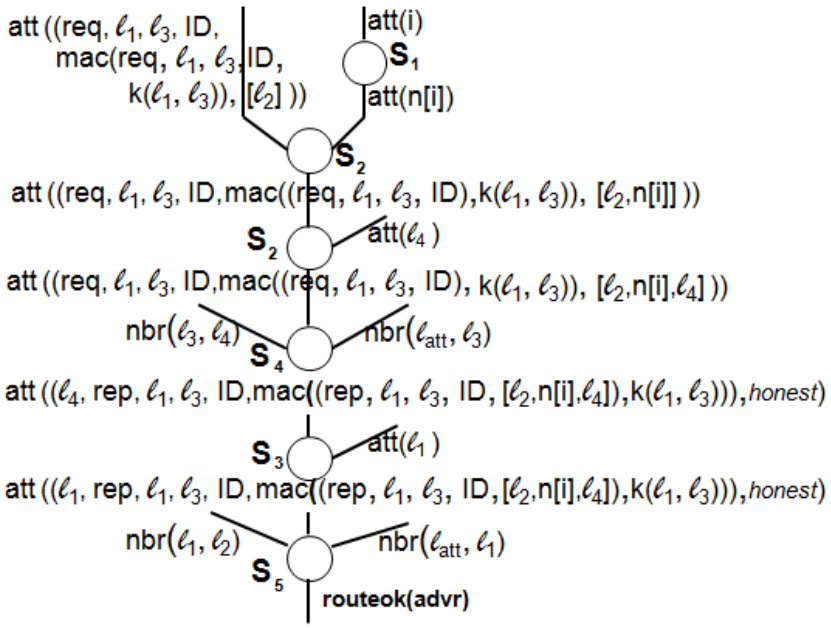


Figure 21: *The behaviour of the attacker node after intercepting the message* $(req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p])$.

## Acknowledgment

The work described in this paper has been supported by the grant TAMOP - 4.2.2.B-10/12010-0009. at the Budapest University of Technology and Economics.

## 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] J. Bengtsson and F. Larsson. Uppaal a tool for automatic verification of real-time systems. *Technical Report,Uppsala University, (96/67)*, 1996.

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

[8] A. Bloch, M. Frederiksen, and B. Haagensen.

[9] L. Buttyán and T. V. Thong. Formal verification of secure ad-hoc network routing protocols using deductive model-checking. In *Proceedings of the IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 1–6, Budapest, Hungary, October 18-20 2010. IFIP.

[10] 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.

[11] C. Fournet and M. Abadi. Mobile values, new names, and secure communication. In *In Proceedings of the 28th ACM Symposium on Principles of Programming, POPL'01*, pages 104–115, 2001.

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

[13] J. C. Godskesen. A calculus for mobile ad-hoc networks with static location binding. *Electron. Notes Theor. Comput. Sci.*, 242(1):161–183, 2009.

[14] G. J. Holzmann. The model checker spin. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 23(5), 1997.

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

[16] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: a secure on-demand routing protocol for ad hoc networks. *Wirel. Netw.*, 11(1-2):21–38, 2005.

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

[18] J. D. Marshall, II, and X. Yuan. An analysis of the secure routing protocol for mobile ad hoc network route discovery: Using intuitive reasoning and formal verification to identify flaws. Technical report, THE FLORIDA STATE UNIVERSITY, 2003.

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

[20] A. Perrig, J. D. Tygar, D. Song, and R. Canetti. Efficient authentication and signing of multicast streams over lossy channels. *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, page 56, 2000.

[21] M. Poturalski, P. Papadimitratos, and J.-P. Hubaux. Towards provable secure neighbor discovery in wireless networks. *Proceedings of the 6th ACM workshop on Formal methods in security engineering*, pages 31–42, 2008.

[22] M. Saksena, O. Wibling, and B. Jonsson. Graph grammar modeling and verification of ad hoc routing protocols. 2008.

[23] 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.

[24] O. Wibling, J. Parrow, and A. Pears. Automatized verification of ad hoc routing protocols. *Formal Techniques for Networked and Distributed Systems FORTE*, pages 343–358, 2004.