Model Checking Ad Hoc Network Routing Protocols: ARAN vs. endairA

Davide Benetti Massimo Merro Luca Viganò
Dipartimento di Informatica, Università degli Studi di Verona, Italy
Email: davbenetti@gmail.com, massimo.merro@univr.it, luca.vigano@univr.it

***Abstract*—Several different secure routing protocols have been proposed for determining the appropriate paths on which data should be transmitted in ad hoc networks. In this paper, we focus on two of the most relevant such protocols, ARAN and endairA, and present the results of a formal analysis that we have carried out using the AVISPA Tool, an automated model checker for the analysis of security protocols. By model checking ARAN with the AVISPA Tool, we have discovered three attacks (a route disruption, a route diversion, and a creation of incorrect routing state), while our analysis of endairA revealed no attacks.**

***Keywords*-Secure routing protocols; ad hoc networks; formal protocol analysis; model checking.**

I. Introduction

Ad hoc networking is a new area[1] in wireless communications that is attracting the attention of many researchers for its potential to provide ubiquitous connectivity without the assistance of any fixed infrastructure. A *Mobile Ad Hoc Network* (MANET) is an autonomous system composed of both *stationary* and *mobile* devices communicating with each other via radio transceivers. While stationary devices cannot move (i.e., their physical location does not vary with time), mobile devices are free to move randomly and organise themselves arbitrarily; thus, the network's wireless topology may change rapidly and *unpredictably*.

Wireless devices use radio frequency channels to *broadcast* messages to the other devices. This form of broadcast is quite different from the more conventional wired-based broadcast as radio transmissions span over a limited area, called transmission *cell*, and reach only a (possibly empty) subset of the devices in the network.

Ad hoc networks rely on multi-hop wireless communications where nodes have essentially two roles: (i) acting as end-systems and (ii) performing routing functions. A routing protocol is used to determine the appropriate paths on which data should be transmitted in a network. Routing protocols for wireless systems can be classified into topology-based protocols and position-based ones:

- *Topology-based protocols* rely on traditional routing concepts, such as maintaining routing tables or distributing link-state information.
- *Position-based protocols* use information about the physical locations of the nodes to route data packets to their destinations.

Topology-based protocols can be further divided into proactive protocols and reactive ones:

- *Proactive routing protocols* try to maintain consistent routing information within the system at any time.
- In *reactive routing protocols*, a route is established between a source and a destination only when it is needed. For this reason, reactive protocols are also called *on-demand* protocols.

Examples of proactive routing protocols for MANETs are OLSR [1] and DSDV [2], while examples of on-demand protocols are DSR [3] and AODV [4].

Initial work on routing in ad hoc networks has considered only the problem of providing efficient mechanisms for finding paths, without considering security issues. However, due to the lack of physical protection, some of the routers could be corrupted affecting the routing paths, which can obviously have undesirable effects on the operations of the network. As a consequence, ad hoc routing protocols are exposed to a number of different attacks [5]. In order to overcome these problems, several "secure" routing protocols have been proposed; for instance, some of the most relevant such protocols for MANETs are: SAODV [6], SRP [7], Ariadne [8], ARAN [9], and endairA [10].

These protocols, like, in general, security protocols, are however notoriously difficult to get right and experience has shown that informal or semi-formal validation approaches are not up to the task of showing a protocol correct. Several approaches have thus been proposed for the formal specification and analysis of security protocols, and a number of efficient tools have been implemented for security protocol *falsification* (i.e., detecting attacks) and/or *verification* (i.e., proving the protocols correct). Two prominent such tools are ProVerif [11] and the AVISPA Tool [12]. However, security protocol analysis has mostly focussed on "standard" protocols for properties such as confidentiality, authentication or key-exchange, and, as we discuss in more detail in the related work section, only few attempts have been made at formally analysing routing protocols.

Contribution: In this paper, we present a formal automated analysis of some of the most common execution scenarios of the "secure" ad hoc routing protocols ARAN

This work was partially supported by the FP7-ICT-2007-1 Project no. 216471 "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" and the PRIN'07 project "SOFT".

[1] Actually, this kind of networks were first studied in the seventies under the name of *Packet Radio Networks*.

and endairA. We have carried out this analysis by using the AVISPA Tool, a state-of-the-art push-button tool for the Automated Validation of Internet Security Protocols and Applications, which provides a modular and expressive formal language for specifying protocols and their security properties, and integrates different back-ends that implement a variety of automatic protocol analysis techniques. All the back-ends of the tool analyse protocols under the assumptions of *perfect cryptography* and that the protocol messages are exchanged over a network that is under the control of a *Dolev-Yao intruder* [13]. Hence, such a model is particularly well suited for the protocols we consider here.

By model checking the two most common execution scenarios of ARAN with the AVISPA Tool, we have discovered the following attacks:

- *route disruption*, which occurs when the intruder prevents a route from being discovered;
- *route diversion*, which occurs when the intruder does not prevent the establishment of routes, but it achieves that some established routes are diverted;
- *creation of incorrect routing state*, which occurs when the intruder jeopardises the routing states in some nodes.

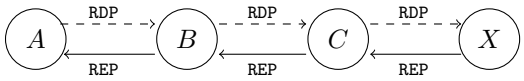
These attacks can be implemented by relying on some *spoofing behaviour* of the intruder. We have found two different kinds of spoofing attacks on ARAN. In the first case, the intruder assumes the identity of a node that has moved away from its initial position; the node remains connected to the rest of the network only because of the intruder. Due to the spoofing activity of the intruder, the node can become part of a routing path, although it is actually disconnected from the rest of the network. This malicious activity can clearly lead to a route-diversion attack as well as a creation-of-incorrect-routing-state attack, as routing tables would contain incorrect information. A different spoofing attack can be achieved by using a number of malicious nodes to immediately forward route requests towards the destination. In this manner, the intruder bypasses the nodes in the route path together with the cryptographic calculations of the protocol. This immediately leads to a route-disruption attack as well as a creation-of-incorrect-routing-state attack.

We have then used the AVISPA Tool to model check the most common execution scenario of the endairA protocol. In this case, our analysis revealed no attacks, the reason being some crucial check on neighbouring nodes imposed by the endairA protocol.

Outline: In Section II, we present ARAN and then, in Section III, we describe endairA by comparison with ARAN. In Section IV, we give a classification of attacks on ad hoc routing protocols, according to [5]. In Section V, we give a brief description of the AVISPA Tool. In Section VI, we present a formal analysis of ARAN using the AVISPA Tool. A similar analysis is done in Section VII for the protocol endairA, showing that endairA is more robust than ARAN. In the final Section VIII, we summarise our main results, compare with related work, and discuss future work. An appendix contains the full specifications of ARAN and endairA that we have formalised for our analysis with the AVISPA Tool.

Figure 1 The ARAN protocol (an example with four nodes)

$$
\begin{array}{lcll}
A & \rightarrow & * & : \{\mathtt{RDP}, X, N_A\}_{K_{A-}}, [cert_A] \\
B & \rightarrow & * & : \{\{\mathtt{RDP}, X, N_A\}_{K_{A-}}\}_{K_{B-}}, [cert_A, cert_B] \\
C & \rightarrow & * & : \{\{\mathtt{RDP}, X, N_A\}_{K_{A-}}\}_{K_{C-}}, [cert_A, cert_C] \\
X & \rightarrow & C & : \{\mathtt{REP}, A, N_A\}_{K_{X-}}, [cert_X] \\
C & \rightarrow & B & : \{\{\mathtt{REP}, A, N_A\}_{K_{X-}}\}_{K_{C-}}, [cert_X, cert_C] \\
B & \rightarrow & A & : \{\{\mathtt{REP}, A, N_A\}_{K_{X-}}\}_{K_{B-}}, [cert_X, cert_B]
\end{array}
$$



II. The ARAN routing protocol

ARAN (Authenticated Routing for Ad Hoc Networks) is a secure, on-demand, distance-vector routing protocol for ad hoc networks proposed in [9]. ARAN uses public key cryptography to ensure routing message integrity and non-repudiation to the route discovery process.

In order to explain in detail how the protocol works, we use some standard notation for security protocols. Given a node n, we write K_{n+} and K_{n-} to denote the public key and the private key of n, respectively. Moreover, given a message $\langle v_1, \ldots, v_n \rangle$, we write $\{v_1, \ldots, v_n\}_{K_{n-}}$ meaning the message $\langle v_1, \ldots, v_n \rangle$ signed by n. In the following, as is standard, we identify the name of a node with its *IP address*. Moreover, for brevity and when no confusion arises, we will often write a message $\langle v_1, \ldots, v_n \rangle$ simply as $v_1, \ldots, v_n$.

ARAN requires the use of a *trusted certificate server* T, whose public key is known to all valid nodes. This server sends to each node a certificate, containing the IP address of the node, its public key, a timestamp t of when the certificate was created, and a time e at which the certificate expires, all signed with the private key of T. As an example, when a node n gets certified by T it receives a certificate of the form $\{n, K_{n+}, t, e\}_{K_{\mathsf{T}-}}$. All nodes use certificates issued by T to authenticate themselves during the protocol.

In Figure 1, we show, as a concrete example, a scheme of the ARAN protocol with four nodes: a *source node* A, a *destination node* X, and two *intermediate nodes* B and C, one close to the source and the other close to the destination. We also provide a graphical representation of the flow of messages: dashed arrows denote the broadcast of *route discovery packets* (RDP), while the continuous arrows denote the unicast sending of *reply packets* (REP). We do not show in the figure the preliminary phase in which each node

receives a certificate from T, assuming it has been already performed. The protocol proceeds as follows.

- The source node A begins the process of discovery of the route to the destination X by broadcasting a route request message of the form $\langle\{\texttt{RDP}, X, N_A\}_{K_{A-}}, [cert_A]\rangle$, where RDP is the packet identifier, X is the IP address of the destination, and N_A is a nonce; all this information is signed with the private key of the source. The purpose of the nonce is to uniquely identify a RDP coming from a source; it also helps in detecting replayed messages since whenever A starts a new route discovery process, it monotonically increases the nonce.
- When a node receives a RDP, it sets an entry in its routing table with A as destination, and the neighbour from which it received the request as next hop. This information will be useful for the reverse path back to the source, when sending a reply message.
- When the intermediate node B receives the request packet, it verifies that the certificate has not expired, and then uses the public key of A, obtained from $cert_A$, to validate the signature. The receiving node also checks the nonce to verify that it has not already processed this RDP: nodes do not forward the same message twice. If any of these verifications fails, the message is dropped, otherwise B signs the received RDP and appends its own certificate $cert_B$. Thus, B broadcasts a packet of the form $\langle\{\{\texttt{RDP}, X, N_A\}_{K_{A-}}\}_{K_{B-}}, [cert_A, cert_B]\rangle$. Hence, in general, the request contains two signatures: that of the source and that of the last intermediate node that processed it.
- When the intermediate node C receives the RDP sent by B, it first verifies if the certificates are still valid, then it validates the signatures for both the source A and the node B using the public keys extracted from the certificates in the message. If any of these verifications fails, the message is dropped, otherwise C records B in its routing table as its parent node. Then, C removes the certificate and the signature of B from the message, it signs the original (signed) message, and appends its own certificate. Thus, C broadcasts a packet of the form $\langle\{\{\texttt{RDP}, X, N_A\}_{K_{A-}}\}_{K_{C-}}, [cert_A, cert_C]\rangle$.
- When the destination X receives the first route request belonging to this route discovery, it performs the verification and updates its routing tables in a manner similar to the intermediate nodes. Then, it sends a REP to the source A passing by the node from which it has received the RDP, i.e., C. The REP is propagated back on the reverse of the discovered route as a unicast message. The destination node X sends to C the message $\langle\{\texttt{REP}, A, N_A\}_{K_{X-}}, [cert_X]\rangle$, which contains the packet type identifier REP, the IP address of the source, and the nonce N_A originally sent by the source, all signed with the private key of the destination, together with the certificate of X.
- When C receives the reply packet from X, it verifies if X's certificate has not expired and validates its signature. If any of these verifications fails, the message is dropped, otherwise C signs the message with its private key, and appends its own certificate before forwarding the packet to its parent node. Thus, C sends to B a message of the form $\langle\{\{\texttt{REP}, A, N_A\}_{K_{X-}}\}_{K_{C-}}, [cert_X, cert_C]\rangle$. Eventually, the message is received by B, which does the usual verifications, removes the signature and the certificate of C, signs the original (signed) message, appends its own certificate, and sends to A the reply message. That is, B sends to the source A a reply message of the form $\langle\{\{\texttt{REP}, A, N_A\}_{K_{X-}}\}_{K_{B-}}, [cert_X, cert_B]\rangle$.
- When A receives the reply, it verifies whether the certificates are valid, validates the signatures, and verifies the nonce returned by the destination. If any of these verifications fails, the message is dropped, otherwise A notices that the intended destination was reached and that B is the first next-hop towards the destination.

As can be seen from the description, ARAN does not use hop counts as a routing metric. Instead, the nodes update their routing tables using the information obtained from the routing message arriving first; any later message that belongs to the same route discovery is discarded. This means that ARAN does not necessarily discover the shortest path in the network, but rather it discovers the quickest one.

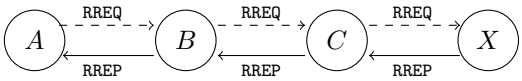
III. The endairA routing protocol

In Figure 2, we show, as a concrete example, a scheme of the endairA protocol with four nodes: a source A, a destination X, and two intermediate nodes B and C, one close to the source and the other close to the destination. Again, we give a graphical representation of the message flow, where the dashed arrows denote the broadcast of *route request packets* (RREQ) and the continuous ones denote the unicast sending of *route reply packets* (RREP). The protocol works as follows.

- The source A broadcasts a RREQ of the form $\langle\texttt{RREQ}, A, X, id, [\,]\rangle$, where id is a randomly generated request identifier that helps in detecting replay messages, and $[\,]$ is the list of intermediate nodes that have successfully received the RREQ. This list is initially empty.
- When the intermediate node B receives the route request for the first time, it broadcasts the message $\langle\texttt{RREQ}, A, X, id, [B]\rangle$, appending its node-id in the list.
- When the intermediate node C receives the route request sent by B, it similarly broadcasts a RREQ of the form $\langle\texttt{RREQ}, A, X, id, [B, C]\rangle$.
- When the request reaches the destination, the node X replies with a unicast message RREP for C

Figure 2 The endairA routing protocol

$$
\begin{array}{lcll}
A & \longrightarrow & * & : \ \texttt{RREQ}, A, X, id, [\ ] \\
B & \longrightarrow & * & : \ \texttt{RREQ}, A, X, id, [B] \\
C & \longrightarrow & * & : \ \texttt{RREQ}, A, X, id, [B, C] \\
X & \longrightarrow & C & : \ \{\texttt{RREP}, A, X, [B, C]\}_{K_{X-}} \\
C & \longrightarrow & B & : \ \{\{\texttt{RREP}, A, X, [B, C]\}_{K_{X-}}\}_{K_{C-}} \\
B & \longrightarrow & A & : \ \{\{\{\texttt{RREP}, A, X, [B, C]\}_{K_{X-}}\}_{K_{C-}}\}_{K_{B-}}
\end{array}
$$



(the last node in the route-list) of the form $\{\texttt{RREP}, A, X, [B, C]\}_{K_{X-}}$.

- When C receives this message, it verifies if the signature is valid and extracts the content of the signed message. Then, it verifies that its node-id is in the node list of the reply, and that the previous identifier, B in this case (or that of the source if there is no previous identifier in the node list), and the following identifier, X in this case, belong to neighbouring nodes. Each intermediate node also verifies the digital signatures of the received reply messages. If any of these verifications fails, then the reply message is dropped. Otherwise, the message is signed by the intermediate node and passed to the next node on the route (towards the source). So, in our case, at the end of the protocol, node B sends to the source A the unicast message $\{\{\{\texttt{RREP}, A, X, [B, C]\}_{K_{X-}}\}_{K_{C-}}\}_{K_{B-}}$.
- When A receives the RREP, it verifies the signatures, extracts the content of the message, and verifies if the first identifier in the route is a neighbour, and finally that the reply message is the expected one. If any of these verifications fails, the reply message is dropped.

IV. Attacks in ad hoc routing protocols

Routing is a fundamental service in any kind of network; hence, an ideal target for attacks. According to [5], attacks against a routing protocol can have the following three goals:

- increase intruder control over the communications between some nodes;
- degrade the quality of the service provided by the network;
- increase the resource consumption of some nodes (e.g., CPU, memory, or energy).

In order to attack an ad hoc routing protocol, an intruder must perform some malicious activities to achieve one or more of the goals described above. These activities include *eavesdropping*, *replaying*, *modifying* and *deleting* control packets (i.e., packets containing routing information). In addition, an intruder can try to *fabricate* control packets containing fake routing information, or it can create control packets under a fake identity: the former is called *packet forgery*, and the latter is usually referred to as *spoofing*. Using these instruments, an intruder can mount the following attacks against routing protocols.

Route disruption: In a route-disruption attack, the intruder prevents a route from being discovered between two nodes that are otherwise connected. In this case, there exists a route between the two victim nodes, but due to the intruder, the routing protocol is unable to discover it. The primary objective of this attack is to degrade the quality of the service provided by the network. In particular, the two victims cannot communicate, and other nodes can also suffer and be coerced to use suboptimal routes. There are several ways to implement a route-disruption attack. For instance, if the intruder controls a set of nodes that form a vertex cut in the network, then it is fairly easy to prevent the discovery of any routes between the two parts of the network by dropping all control packets sent from one part to the other. Another way to mount a route-disruption attack is to forge error messages that would invalidate the correct routing state in some victim nodes, thereby effectively preventing them from being able to communicate with some other nodes.

Route diversion: In a route-diversion attack, the intruder does not prevent the establishment of routes, but it achieves that some established routes are diverted. This means that due to the presence of the intruder, the protocol establishes routes that are different from those that it would establish if the intruder did not interfere with the execution of the protocol. The goal of route diversion can be to increase adversarial control over the communications between some victims nodes. In this case, the intruder tries to achieve that the diverted routes contain one of the nodes that it controls or a link that it can observe. Then, the intruder can more easily eavesdrop or modify data sent between the victim nodes. Another objective of route diversion can be to increase the resource consumption of some nodes. Route diversion can be implemented by dropping, forging or manipulating control packets.

Creation of incorrect routing state: Another type of attack aims at jeopardising the routing state in some nodes so that the state appears to be correct but, in fact, it is not and thus data packets routed using that state will never reach their destination. One example of this attack is when the route discovery procedure of a source routing protocol returns a non-existent route to the source; as a consequence, data packets using this non-existent route will be dropped when they reach the first non-existent link in the route. Another example is the creation of routing loops. Distance-vector-based protocols are particularly vulnerable to this kind of attack, because the nodes do not have a full view of the whole network or of the entire route. Yet another example is when the network is disconnected, but the routing state in some node falsely indicates that each destination

is reachable. Incorrect routing states can be created by spoofing, forging, modifying, or dropping control packets.

Generation of extra traffic: An attack aiming at increasing resource consumption can inject spoofed packets into the network. In on-demand protocols, a spoofed route request can be flooded in this way. In distance-vector-based protocols, a spoofed routing update message can cause a sequence of "triggered" updates propagating in the whole network.

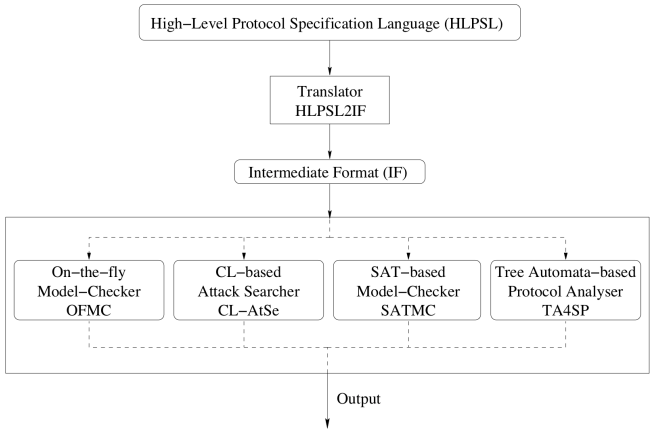
Setting up a gray hole: All the attacks that we have discussed so far target the route establishment function of routing, whereas a gray-hole attack is concerned with the packet forwarding function. In a gray-hole attack, an intruder selectively drops data packets that it should forward. If all data packets are dropped, then the gray hole degenerates into a black hole. The primary objective of the gray-hole attack is to degrade the quality of the service. In particular, the packet delivery ratio between some nodes can decrease considerably.

V. The AVISPA Tool

The AVISPA Tool consists of different modules, interconnected as shown in Figure 3; for more details see the documentation and papers listed at [12]. A protocol designer interacts with the tool by specifying a security problem (a protocol paired with one or more security properties that the protocol is expected to achieve) in the *High-Level Protocol Specification Language HLPSL* [14]. The HLPSL is an expressive, modular, role-based, formal language that allows for the specification of control flow patterns, data-structures, different intruder models, complex security properties, as well as different cryptographic primitives and their algebraic properties. These features make HLPSL well suited for specifying industrial-scale protocols and it has been applied for the specification of dozens of such protocols and for their analysis using the AVISPA Tool. Moreover, HLPSL is also well suited for the specification (and thus AVISPA is also well suited for the analysis) of wireless sensor network security protocols, as in [15], or of secure routing protocols for ad hoc networks, as we do here.

The AVISPA Tool also comprises a more low-level, rewrite-based specification language, called *Intermediate Format IF*. Specifications of security problems written in HLPSL are automatically translated into IF specifications, which describe infinite-state transition systems amenable to formal analysis. IF specifications are then automatically given as input to the different back-ends of the AVISPA Tool, which implement a variety of automatic analysis techniques ranging from *protocol falsification* by finding an attack on the input protocol and *bounded-session verification* (for a given scenario consisting of a finite number of sessions), to *abstraction-based unbounded verification* methods for infinite numbers of sessions.

Figure 3 Architecture of the AVISPA Tool



The tool integrates four back-ends:[2] the *On-the-fly Model-Checker OFMC*, the *Constraint-Logic-based Attack Searcher CL-AtSe*, the *SAT-based Model-Checker SATMC*, and the *TA4SP protocol analyser*, which verifies secrecy properties of protocols by implementing tree automata based on automatic approximations. All the back-ends of the tool analyse protocols under the assumptions of *perfect cryptography* and that the protocol messages are exchanged over a network that is under the control of a *Dolev-Yao intruder* [13]. That is, the back-ends analyse protocols by considering the standard protocol-independent, asynchronous model of an active intruder who controls the network but cannot break cryptography; in particular, the intruder can intercept messages and analyse them if he possesses the corresponding keys for decryption, and he can generate messages from his knowledge and send them under any party name.

Upon termination, each back-end of the AVISPA Tool outputs the result of its analysis stating whether the input problem was solved (giving a description of the considered protocol goal or, in case it was violated, the related attack trace), some of the system resources were exhausted, or the problem was not tackled by a back-end for some reason.

In the case of our analyses of ARAN and endairA, as we will see in the next two sections, the AVISPA Tool has not revealed any attacks on endairA but it has discovered three attacks on ARAN by means of the back-ends OFMC, CL-AtSe and SATMC, while TA4SP could not be applied in the first place as it only verifies secrecy properties, which are not the kinds of properties we were interested in here.

Before we discuss our analyses, it is important to observe that in this paper we only consider some of the most

[2]The AVISPA Tool is also available as part of the AVANTSSAR Platform at www.avantssar.eu, including the *Open source Fixedpoint Model-Checker* [16] as the new OFMC. Moreover, the AVISPA Tool is additionally supported by the Security Protocol Animator SPAN, which allows a modeller to interactively build Message Sequence Charts of the protocol execution from HLPSL specifications.

common execution scenarios (and corresponding network topologies) of ARAN and endairA, consisting of a finite number of protocol sessions executed in parallel. (This does not mean, however, that the resulting model-checking problems are finite, as the Dolev-Yao intruder can send infinitely many different messages, which can be dealt with by the AVISPA back-ends.) Hence, this work constitutes a stepping stone for future analyses, in which we will consider further scenarios and sessions, exploiting in particular the symbolic sessions feature of AVISPA (that allows the user to specify simply the number of sessions that the user wishes to analyse) as well as the abstraction techniques that allow for unbounded protocol verification instead of the protocol falsification and bounded-session verification we consider here.

VI. Analysis of ARAN

In order to analyse the "secure" routing protocol ARAN, we have formalised two HLPSL specifications, representing *two different network topologies that correspond to the most common execution scenarios* of ARAN; the specifications are given in full in the appendix. The AVISPA Tool has found a different spoofing attack in each such topology, giving rise to route disruption, route diversion and creation-of-incorrect-routing-state attacks.

A. First HLPSL specification of ARAN

The first HLPSL specification formalises a standard model of ARAN, in which there are four protocol *roles* that can be considered as equivalence classes between nodes with the same behaviour for sending and receiving messages in a generic route request: a source node A, a destination node X, and two intermediate nodes B and C, where B communicates with the neighbouring A and C, and C communicates with the neighbouring B and X. We have chosen to consider two different intermediate nodes of the route discovery because during the request the neighbours of the source node A act differently than the others, as they decrypt and analyse only the private key and the certificate of A; in contrast, the other intermediate nodes must verify two keys and certificates. In the reply, the situation is dual as the node near the destination X in the path receives a single private key and certificate.

The properties of this specification that we have to analyse are the following:

- *Authentication of N_A and X in the request phase*: the receiving node checks (N_A, X) to verify that it has not already processed this RDP.
- *Authentication of N_A and A in the reply phase*: the receiving node checks (N_A, A) to verify that it has not already processed this REP.

The authentication is end-to-end, checked by each node each time it receives a request or a reply.

Our HLPSL specification formalises all this as follows. First of all, it simulates both the request phase and the reply phase: the source node A starts the session and sends a message that reaches the destination X through two intermediate nodes B and C; similarly, X sends a reply that reaches A through B and C. In general, HLPSL specifications are written under the assumption that protocol models have behaviours that can be expressed as *linearly ordered traces of events*, e.g., sending and receiving of messages. In addition, traces contain *auxiliary events* that express information about an honest principal's assumptions or intentions when executing a protocol.[3] These events provide a language over which we then define the goals of the protocol, where we assume that the intruder can neither generate auxiliary events nor modify those auxiliary events generated by honest principals. In particular, for authentication, one can use the auxiliary events `witness` and `request` to check that a principal is right in believing that its intended peer is present in the current session, has reached a certain state, and agrees on a certain value, which typically is fresh. Note that, here and in the following, we use the `tt` font for actual HLPSL code.

There is also an auxiliary event `wrequest` that corresponds to *weak authentication*, i.e., authentication without replay protection, which is the kind of authentication we are interested in here. In a nutshell:

- `witness(A,B,v,M)` should be read as "principal `A` asserts that it wants to be the peer of principal `B`, agreeing on the value `M` in an authentication effort identified by the protocol id `v`."
- `wrequest(B,A,v,M)` should be read as "principal `B` accepts the value `M` and now relies on the guarantee that principal `A` existed sometime in the past and at that time agreed on value `M`, having interpreted it as protocol id `v`."

The third argument `v` is used for distinguishing different authentication pairs; that is, for asserting with what purpose the value is being interpreted. As a modelling convention, it is usually the name of the authenticating role, the role to be authenticated, and the name of the variable being checked, all in lower case, concatenated together.

In the goal section of the protocol specification, we then write, for instance, `weak_authentication_on b_a_M` to indicate that `witness`es and `wrequest`s containing those three protocol ids should be taken into account.

For concreteness, in our first HLPSL specification of ARAN, we consider weak authentication as obtained by placing a pair `wrequest` and `witness` of the variables `X` and `Na` in the request phase, and of the variables `A` and `Na` in the reply phase (where `Na` is the HLPSL representation of

[3] As is standard, we use the term *principal* to refer to an agent/node participating in a protocol, and call it *honest* when it behaves only according to what specified by the protocol.

nonce N_A). More precisely, the source node `A` contains the `witness` with `B` of `X` and `Na` in the request phase and the `wrequest` with `B` of `A` and `Na` in the reply phase. The first intermediate node `B` contains both the `wrequest` and the `witness` with `A` and `C` of `X` and `Na` in the request phase, and `A` and `Na` in the reply phase. The second intermediate node `C` contains both the `wrequest` and the `witness` with `B` and `X` of `X` and `Na` in the request phase, and `A` and `Na` in the reply phase. The destination node `X` contains the `wrequest` with `C` of `X` and `Na` in the request phase, and the `witness` with `B` of `A` and `Na` in the reply phase.

In addition to the four basic roles we have discussed, our HLPSL specification contains also a composed role `session` that describes one whole protocol session by instantiating one instance of each basic role, "glueing" them so they execute together, usually in parallel (with interleaving semantics). Finally, we have a top-level role `environment` that contains global constants and a composition of one or more sessions, where the intruder may play some roles as a legitimate user; this is, in other words, the scenario we are analysing. There is also a statement that describes what knowledge the intruder initially has; typically, this includes the names of all principals, all public keys, his own private key(s), and all publicly known functions. Note that the constant `i` is used to refer to the intruder.

B. Analysis of the first HLPSL specification of ARAN

The analysis with the AVISPA Tool of our first HLPSL specification of ARAN finds a spoofing attack, where the intruder pretends to be an intermediate node or even the destination node, thereby deceiving the other nodes:

1) $A \rightarrow I \ : \ \{RDP, X, N_A\}_{K_{A-}}, cert_A$
2) $I \rightarrow B \ : \ \{RDP, X, N_A\}_{K_{A-}}, cert_A$
3) $B \rightarrow I \ : \ \{\{RDP, X, N_A\}_{K_{A-}}\}_{K_{B-}}, cert_A, cert_B$

Figure 4 shows the network topology, where A represents the source node, B represents the intermediate node, neighbour to A, and I represents the intruder.

When, in a network topology, an intermediate node B moves from his initial position to the next one, outside of the communication range of the other nodes in the network, the intruder can place himself within the ranges of both A and B. Now, via the spoofing, a false routing information attack may occur. A broadcasts the route request, as specified by the protocol; I intercepts this message but cannot send it pretending to be B, because it doesn't have the private key of B to sign the packet. So, I relays the request to B, who thinks that I is A and re-broadcasts the message. However, only I can overhear this message and sends it to the next intermediate node C. In this way, if this packet reaches the destination the first time around, it will establish an incorrect route state. Obviously, it is necessary that the same procedure is carried out in the reply phase, where there are similar controls. After the intruder executes the sequence, it can thus realise an incorrect-routing-state attack.

Figure 4 Network topology of the first attack on ARAN

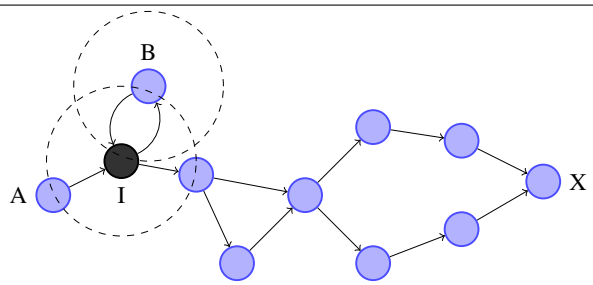
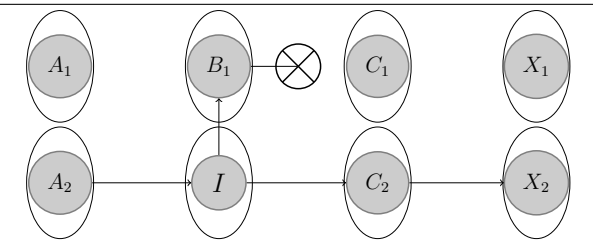


Figure 5 Equivalence classes in the first HLPSL specification of ARAN



As illustrated in Figure 5, our HLPSL specification contains two parallel sessions, which, for readability, we denote by (A_1, B_1, C_1, X_1) and (A_2, I, C_2, X_2), where the indexing is used only to distinguish the two sessions executed by the same principals (e.g. A_1 and A_2 are both A): in the first session all four basic roles are played by honest principals, while in the second session the intruder plays the role of B_2. More specifically, in the second session A_2 starts the route discovery by sending a request to I, but I does not possess the private key of B to sign the message, and thus cannot fully take over B's identity. So, I forwards the message to B_1 and, according to the first session of the protocol, B_1 sends the new signed message to C_1. But I can intercept this message and, afterwards, relay the packet to C_2 in the second session of the protocol. Since the broadcast actually makes it impossible to block messages in a MANET, we have modelled this in our network by considering a node that moves outside the communication range. The same situation must of course also occur in the reply phase, in which C_2 will send a message to I, who will follow the same procedure to make it reach the source A_2, properly signed by B_1. This attack can also occur when in the second session I plays the role of the second intermediate node C_2 or of the destination X_2.

C. Second HLPSL specification of ARAN and its analysis

By considering another common execution scenario, we have discovered a second spoofing attack on ARAN, whose network topology is illustrated in Figure 6. It is namely possible that a chain of collaborating intruders develop a parallel path from the source node A to the destination X. Thus, since the intruders aren't required to perform

Figure 6 Network topology of the second attack on ARAN

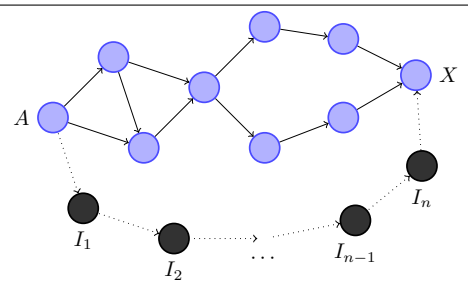
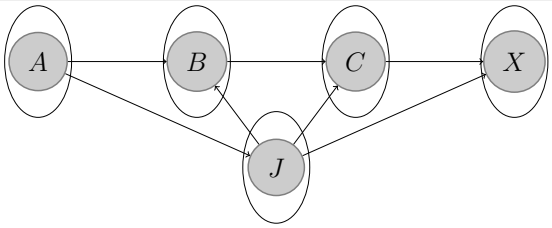


Figure 7 Equivalence classes in the second HLPSL specification of ARAN



cryptographic operations, the request messages may reach the destination X faster than in the authorised network. Naturally, this operation can be done within any portion of the network. In the reply, the intruders execute the same operations "back to front". It is thus an attack of route disruption, since the nodes of the network don't know the intruders I_1, I_2, ..., I_n.

To model this attack, we have formalised a second HLPSL specification, in which, as illustrated in Figure 7, we added a particular polymorphed role, which we call *joker role* J and which includes all the paths that reach any of the following node equivalence classes. So, J feigns the normal execution of the protocol to simulate the chain of intruders (in fact, the presence of J is just a technicality that allows us to avoid having to specify the chain of communicating intruders).

The attack proceeds as follows:

1) $A \rightarrow I \ : \ \{RDP, X, N_A\}_{K_{A-}}, cert_A$
2) $I \rightarrow J \ : \ \{RDP, X, N_A\}_{K_{A-}}, cert_A$
3) $J \rightarrow I \ : \ \{\{RDP, X, N_A\}_{K_{A-}}\}_{K_{C-}}, cert_A, cert_C$
4) $I \rightarrow X \ : \ \{\{RDP, X, N_A\}_{K_{A-}}\}_{K_{C-}}, cert_A, cert_C$
5) $X \rightarrow I \ : \ \{\{REP, A, N_A\}_{K_{X-}}, cert_X$

Specifically, when I receives a message from A, it starts the attack by making use of J to simulate the chain of intruders. J sends an identical packet at the same time to B, C and X; when J receives a message from B, it sends an identical packet at the same time to C and X; when J receives a message from C, it sends an identical packet to X. If, finally, J receives a message from X, it promptly drops the packet. X receives a message signed with the private key of C, because C doesn't represent a single node but rather an equivalence class of all nodes neighbouring the destination, and it is important to have a perfect match between the send channel and the receive channel. Similar to the previous attack, now the intruder can realise an incorrect-routing-state attack or a disruption attack.

Since this specification gives rise to a huge expansion of states of execution, we have actually formalised it to direct the flow of execution towards J and removed the flow towards the other nodes. For this reason, we have introduced a new variable (called `Plus`) encrypted with the private key of J. This minor modification in the protocol is not in contrast with its properties but it allows for a considerable simplification of the whole analysis procedure, and thus leads to the discovery of the attack.

VII. Analysis of endairA

The analysis of endairA uses a variant of the first HLPSL specification of ARAN where, instead of four roles, we have only three roles as all intermediate nodes behave in the same way. The authentication is end-to-end, as in ARAN, so the properties we have to analyse are the authentication of A, B and X in the reply phase. Each intermediate node that receives the reply verifies that its identifier is in the node list included in the reply and that the preceding node identifier (or that of the initiator, if there is no preceding identifier in the node list) and the following node identifier (or that of the target, if there is no following identifier in the node list) indeed belong to neighbouring nodes. Each intermediate node also verifies that the digital signatures in the reply are valid and that they correspond to the subsequent identifiers in the node list and to the target. When the source node receives the route reply, it verifies if the first identifier in the route included in the reply belongs to a neighbour.

We have used AVISPA only to analyse the reply phase since endairA focusses on this phase, but does not really attempt at protecting the request phase so to reduce the use of cryptography and thus, ultimately, to reduce the resource consumption in the nodes.

As described above, endairA adds a property with respect to ARAN, namely the check of the node identifiers in the node list in the reply phase. It is because of this additional check that AVISPA does not find an attack similar to that of ARAN for endairA: we have modelled this property by specifying, in the transitions, that whenever a node receives a new message, it checks that its identifier and that of the neighbouring nodes do indeed correspond to what he initially received from the role `session`. If an intruder attempted to spoof the identity of another node, in a manner similar to what we saw before, the following node would notice the problem and interrupt the execution, as the check would fail given that he wouldn't find the appropriate node in his neighbourhood. More concretely, in our HLPSL specification, this is formalised by requiring that a variable remains equal, i.e., that a variable and its primed version (representing its new value) are equal. For example, for the

destination node X, $B = B'$ and $X = X'$. In fact, each node controls his neighbourhood, so the intermediate node B has three equalities, $A = A'$, $B = B'$ and $X = X'$, while the source node A has $A = A'$ and $B = B'$.

The AVISPA Tool has validated our specification, which formalises the most common execution scenario of endairA. This, of course, does not mean that endairA is free from attacks and we are currently formalising and analysing other interesting scenarios.

VIII. Conclusions, Related and Future Work

In this paper, we have presented a formal approach to the security analysis of two ad hoc routing protocols, ARAN and endairA, by means of the AVISPA Tool. We have discovered three kinds of attacks on ARAN: route disruption, route diversion, and creation of an incorrect routing state. In contrast, endairA is more robust, due to the extra check on the neighbouring nodes belonging to the discovered route. As a consequence, our analysis of endairA revealed no attacks.

Attacks against ad hoc routing protocols can be subtle and thus difficult to discover. Since proving that a routing protocol is free from attacks is impossible by informal reasoning (as is the case for security protocol analysis in general, due to the virtually infinite protocol execution scenarios that should be considered), much effort has been recently devoted to modelling and analysing wireless communication and/or routing protocols in a formal way. For example, Yang and Baras [17] provide a symbolic model for routing protocols based on strand spaces, modelling the network topology but not the cryptographic primitives that can be used for securing communications; they also implement a semi-decision procedure to search for attacks.

Buttyán and Vadja [18] provide a model for routing protocols in a cryptographic setting; they provide a security proof (by hand) for a fixed protocol. Ács and Buttyán [19] use a similar cryptographic model for the security analysis of on-demand, distance-vector routing protocols, such as AODV SAODV, and ARAN. In particular, the ARAN protocol was (supposedly) proved secure using this framework. The same authors propose a new framework [20] for on-demand source routing and on-demand distance-vector protocols. In addition, routing security is defined in terms of resistance against attacks aimed at creating an incorrect routing state in the network. Thus, route-disruption, route-diversion, generation of-extra-control-traffic, and gray-hole attacks are not considered within the framework. The protocol endairA has been verified in [20], but note that both frameworks proposed by Ács and Buttán cannot be automated.

Nanz and Hankin [21] propose a process calculus to model the network topology and broadcast communications. They also propose a decision procedure for an intruder that is already specified by the user. This allows them only to check security against fixed scenarios known in advance.

In [22], Godskesen studies a simplification of the ARAN protocol with only two nodes: the initiator and the destination, in the transmission range of each other. The intruder simply relays messages from one node to the other. The specification of the protocol is given in a process calculus for MANETs; the tool ProVerif [11] is then used to show that false routes can be constructed by an intruder establishing a man-in-the-middle spoofing attack. Our work is inspired by and generalises the work done in [22], in the attempt of carrying out an exhaustive analysis of possible attacks on ARAN using AVISPA. To our knowledge, our paper is the first one to use AVISPA for the analysis of ad hoc routing protocols. The AVISPA tool has already been used in [15] for an analysis TinySec [23], LEAP [24], and TinyPK [25], three wireless sensor network security protocols, and in [26] for an analysis of the Sensor Network Encryption Protocol SNEP [27]. Moreover, AVISPA has also been used to check new security properties in ad hoc networks for detecting a particular trust relation between two nodes in an anonymous and unobservable way [28].

More recently, Arnaud, Cortier and Delaune [29] have proposed a calculus for modelling and reasoning about security protocols, including secure routing protocols, for a bounded number of sessions. They provide two NPTIME decision procedures for analysing routing protocols for any network topology, and apply their framework to analyse the protocol SRP [7] applied to DSR [3].

We see this paper as a first step in a research program aiming at applying the AVISPA Tool for the formal analysis of ad hoc network routing protocols. As future work, we thus envisage to use AVISPA both to consider other analysis scenarios of ARAN and endairA (and possibly also attempt at verifying their security and not just find attacks on them) and to analyse other protocols, all also with respect to other kinds of attacks such as extra traffic and gray holes. We also plan to employ the AVANTSSAR Platform for such analyses; the platform, which is currently under development, scales up (the main components of) the AVISPA Tool to the formal specification and automated validation of trust and security of service-oriented architectures, and we thus expect that it will allow us to carry out even more detailed analyses of the protocols under consideration.

References

[1] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," 2003, rFC 3626.

[2] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," in *Proceedings of SIGCOMM*, 1994, pp. 234–244.

[3] D. B. Johnson and D. A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*. Kluwer Acad. Pub., 1996.

[4] C. E. Perkins and E. M. Belding-Royer, "Ad-hoc On-Demand Distance Vector Routing," in *Proceedings of WMCSA*. IEEE Computer Society Press, 1999, pp. 90–100.

[5] L. Buttyan and J. Hubaux, *Security and Cooperation in Wireless Networks*. Cambridge University Press, 2008.

[6] M. G. Zapata and N. Asokan, "Securing Ad Hoc Routing Protocols," in *Proceedings of WiSe*. ACM, 2002, pp. 1–10.

[7] P. Papadimitratos and Z. J. Haas, "Secure Link State Routing for Mobile Ad Hoc Networks," in *Proceedings of SAINT'03 Workshops*. ACM, 2003, pp. 379–383.

[8] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks," *Wireless Networks*, vol. 11, no. 1-2, pp. 21–38, 2005.

[9] K. Sanzgiri, D. LaFlamme, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer, "Authenticated Routing for Ad Hoc Networks," *IEEE Journal on Selected Areas in Communication, special issue on Wireless Ad Hoc Networks*, vol. 23, no. 3, pp. 598–610, 2005.

[10] G. Ács, L. Buttyán, and I. Vajda, "Provably Secure On-Demand Source Routing in Mobile Ad Hoc Networks," *IEEE Transaction on Mobile Computing*, vol. 5, no. 11, pp. 1533–1546, 2006.

[11] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proceedings of CSF'01*. IEEE Computer Society Press, 2001, pp. 82–96.

[12] "The AVISPA Tool," http://www.avispa-project.org/.

[13] D. Dolev and A. Yao, "On the Security of Public-Key Protocols," *IEEE Trans. on Info. Theory*, vol. 2, no. 29, 1983.

[14] Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron, *A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols*, ser. Automated Software Engineering. Austrian Computer Society, 2004, vol. 180, pp. 193–205.

[15] M. L. Tobarra, D. Cazorla, F. Cuartero, G. Díaz, and M.-E. Cambronero, "Model Checking Wireless Sensor Network Security Protocols: TinySec + LEAP + TinyPK," *Telecommunication Systems*, vol. 40, no. 3-4, pp. 91–99, 2009.

[16] S. Mödersheim and L. Viganò, "The Open-Source Fixed-Point Model Checker for Symbolic Analysis of Security Protocols," in *FOSAD 2008/2009*, ser. LNCS 5705. Springer-Verlag, 2009, pp. 166–194.

[17] S. Yang and J. S. Baras, "Modeling vulnerabilities of ad hoc routing protocols," in *Proceedings of SASN*. ACM, 2003, pp. 12–20.

[18] L. Buttyán and I. Vajda, "Towards provable security for ad hoc routing protocols," in *Proceedings of SASN*. ACM, 2004, pp. 94–105.

[19] G. Ács, L. Buttyán, and I. Vajda, "Provable security of on-demand distance vector routing in wireless ad hoc networks," in *Proceedings of ESAS*, ser. LNCS 3813. Springer, 2005, pp. 113–127.

[20] ——, "Provably secure on-demand source routing in mobile ad hoc networks," *IEEE Trans. Mob. Comput.*, vol. 5, no. 11, pp. 1533–1546, 2006.

[21] S. Nanz and C. Hankin, "A Framework for Security Analysis of Mobile Wireless Networks," *Theoretical Computer Science*, vol. 367, no. 1-2, pp. 203–227, 2006.

[22] J. Godskesen, "A Calculus for Mobile Ad Hoc Networks," in *Proceedings of Coordination'97*, ser. LNCS, vol. 4467. Springer-Verlag, 2007, pp. 132–150.

[23] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: a link layer security architecture for wireless sensor networks," in *Proceedings of SenSys'04*. ACM, 2004, pp. 162–175.

[24] S. Zhu, S. Setia, and S. Jajodia, "Leap - efficient security mechanisms for large-scale distributed sensor networks," in *Proceedings of SenSys*. ACM, 2003, pp. 308–309.

[25] R. J. Watro, D. Kong, S. fen Cuti, C. Gardiner, C. Lynn, and P. Kruus, "Tinypk: securing sensor networks with public key technology," in *Proceedings of SASN*, 2004, pp. 59–64.

[26] L. Tobarra, D. Cazorla, and F. Cuartero, "Formal Analysis of Sensor Network Encryption Protocol (SNEP)," in *Proceedings of WSNS 2007*. IEEE CS, 2007.

[27] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "Spins: Security Protocols for Sensor Networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, 2002.

[28] M. Heen, G. Guette, and T. Genet, "On the Unobservability of a Trust relation in Mobile Ad Hoc Networks," in *Proceedings of WISTP'09*, ser. Lecture Notes in Computer Science, vol. 5746. Springer, 2009, pp. 1–11.

[29] M. Arnaud, V. Cortier, and S. Delaune, "Modeling and Verifying Ad Hoc Routing Protocols," in *Proceedings of CSF'10*. IEEE Computer Society Press, 2010.

APPENDIX

FIRST HLPSL SPECIFICATION OF ARAN

```
role source_node(
  A,B,C,X : agent,
  Ka, Kb, Kc, Kt, Kx : public_key,
  RCV, SND : channel(dy))
played_by A def=
  local
    RDP,REP : protocol_id,
    State : nat,
    NA,Ts1,E1,Ts2,E2,Ts3,E3,Ts4,E4 : text
  const
    a_b_IPa, a_b_IPx, a_a_Na : protocol_id
  init State := 0
  transition
    step1.
      State = 0 /\ RCV(start)
      =|>
      State':= 4 /\ Na' := new()
                 /\ SND({RDP.X.Na'}_inv(Ka).{A.Ka.Ts1.E1}_inv(Kt))
                 /\ witness(A,B,a_b_IPx,X) /\ witness(A,B,a_a_Na,Na')
    step2.
      State = 4 /\ RCV({{REP.A'.Na'}_inv(Kx)}_inv(Kb).{X'.Kx.Ts4'.E4'}_inv(Kt).
                   {B'.Kb.Ts2'.E2'}_inv(Kt))
      =|>
      State':= 8 /\ wrequest(A,B,a_b_IPa,A') /\ wrequest(A,B,a_a_Na,Na')
end role

role intermediate_node1(
  A,B,C,X : agent,
  Ka, Kb, Kc, Kt, Kx : public_key,
  RCV,SND : channel(dy))
played_by B def=
  local
    RDP,REP : protocol_id,
    State : nat,
    Na,Ts1,E1,Ts2,E2,Ts3,E3,Ts4,E4 : text
```

SECOND HLPSL SPECIFICATION OF ARAN

```
  const
    a_b_IPa, a_b_IPx, a_a_Na : protocol_id
  init State := 1
  transition
    step1.
      State = 1 /\ RCV({RDP.X'.Na'}_inv(Ka).{A'.Ka.Ts1'.E1'}_inv(Kt))
      =|>
      State':= 5 /\ SND({{RDP.X'.Na'}_inv(Ka)}_inv(Kb).
                     {A'.Ka.Ts1'.E1'}_inv(Kt).{B.Kb.Ts2.E2}_inv(Kt))
                 /\ wrequest(B,A,a_b_IPx,X') /\ wrequest(B,A,a_a_Na,Na')
                 /\ witness(B,C,a_b_IPx,X') /\ witness(B,C,a_a_Na,Na')
    step2.
      State = 5 /\ RCV({{REP.A'.Na'}_inv(Kx)}_inv(Kc).
                   {X'.Kx.Ts4'.E4'}_inv(Kt).
                   {C'.Kc.Ts3'.E3'}_inv(Kt))
      =|>
      State':= 9 /\ SND({{REP.A'.Na'}_inv(Kx)}_inv(Kb).
                     {X'.Kx.Ts4'.E4'}_inv(Kt).{B.Kb.Ts2.E2}_inv(Kt))
                 /\ wrequest(B,C,a_b_IPa,A') /\ wrequest(B,C,a_a_Na,Na')
                 /\ witness(B,A,a_b_IPa,A') /\ witness(B,A,a_a_Na,Na')
end role

role intermediate_node2(
  A,B,C,X : agent,
  Ka, Kb, Kc, Kt, Kx : public_key,
  RCV,SND : channel(dy))
played_by C def=
  local
    RDP,REP : protocol_id,
    State : nat,
    Na,Ts1,E1,Ts2,E2,Ts3,E3,Ts4,E4 : text
  const
    a_b_IPa, a_b_IPx, a_a_Na : protocol_id
  init State := 2
  transition
    step1.
      State = 2 /\ RCV({{RDP.X'.Na'}_inv(Ka)}_inv(Kb).
                   {A'.Ka.Ts1'.E1'}_inv(Kt).{B'.Kb.Ts2'.E2'}_inv(Kt))
      =|>
      State' := 6 /\ SND({{RDP.X'.Na'}_inv(Ka)}_inv(Kc).
                     {A'.Ka.Ts1'.E1'}_inv(Kt).{C.Kc.Ts3.E3}_inv(Kt))
                  /\ wrequest(C,B,a_b_IPx,X') /\ wrequest(C,B,a_a_Na,Na')
                  /\ witness(C,X,a_b_IPx,X') /\ witness(C,X,a_a_Na,Na')
    step2.
      State = 6 /\ RCV({REP.A'.Na'}_inv(Kx).{X'.Kx.Ts4'.E4'}_inv(Kt))
      =|>
      State':= 10 /\ SND({{REP.A'.Na'}_inv(Kx)}_inv(Kc).
                     {X'.Kx.Ts4'.E4'}_inv(Kt).{C.Kc.Ts3.E3}_inv(Kt))
                  /\ wrequest(C,X,a_b_IPa,A') /\ wrequest(C,X,a_a_Na,Na')
                  /\ witness(C,B,a_b_IPa,A') /\ witness(C,B,a_a_Na,Na')
end role

role final_node(
  A,B,C,X : agent,
  Ka, Kb, Kc, Kt, Kx : public_key,
  RCV,SND : channel(dy))
played_by X def=
  local
    RDP,REP : protocol_id,
    State : nat,
    Na,Ts1,E1,Ts2,E2,Ts3,E3,Ts4,E4 : text
  const
    a_b_IPa, a_b_IPx, a_a_Na : protocol_id
  init State := 3
  transition
    step1.
      State = 3 /\ RCV({{RDP.X'.Na'}_inv(Ka)}_inv(Kc).
                   {A'.Ka.Ts1'.E1'}_inv(Kt).{C'.Kc.Ts3'.E3'}_inv(Kt))
      =|>
      State':= 7 /\ SND({REP.A'.Na'}_inv(Kx).{X.Kx.Ts4.E4}_inv(Kt))
                 /\ wrequest(X,C,a_b_IPx,X') /\ wrequest(X,C,a_a_Na,Na')
                 /\ witness(X,C,a_b_IPa,A') /\ witness(X,C,a_a_Na,Na')
end role

role session(
  A,B,C,X : agent,
  Ka, Kb, Kc, Kt, Kx : public_key)
  def=
  local
    RCV1,SND1,RCV2,SND2,RCV3,SND3,RCV4,SND4 : channel(dy)
  composition
    source_node(A,B,C,X,Ka,Kb,Kc,Kt,Kx,RCV1,SND1)
    /\ intermediate_node1(A,B,C,X,Ka,Kb,Kc,Kt,Kx,RCV2,SND2)
    /\ intermediate_node2(A,B,C,X,Ka,Kb,Kc,Kt,Kx,RCV3,SND3)
    /\ final_node(A,B,C,X,Ka,Kb,Kc,Kt,Kx,RCV4,SND4)
end role

role environment()
  def=
  const
    a_b_IPa, a_b_IPx, a_a_Na : protocol_id,
    a,b,c,x : agent,
    ka,kb,kc,kt,kx : public_key
    intruder_knowledge ={a,b,c,x,ka,kb,kc,kt,kx}
  composition
    session(a,b,c,x,ka,kb,kc,kt,kx)
     /\ session(a,i,c,x,ka,kb,kc,kt,kx)
end role

goal
    weak_authentication_on a_b_IPa
    weak_authentication_on a_b_IPx
    weak_authentication_on a_a_Na
end goal

environment()
```

```
role source_node(
  A,B,C,J,X : agent,
  Ka, Kb, Kc, Kj, Kt, Kx : public_key,
  RCV, SND : channel(dy))
played_by A def=
  local
    RDP,REP : protocol_id,
    State : nat,
    Na,Plus,Ts1,E1,Ts2,E2,Ts3,E3,Ts4,E4 : text
  const a_b_IPa, a_b_IPx, a_a_Na : protocol_id
  init State := 0
  transition
    step1.
      State = 0 /\ RCV(start)
      =|>
      State':= 5 /\ Na' := new()
                 /\ SND({RDP.X.Na'}_inv(Ka).{A.Ka.Ts1.E1}_inv(Kt))
                 /\ witness(A,B,a_b_IPx,X) /\ witness(A,B,a_a_Na,Na')
    step2.
      State = 5 /\ RCV({{REP.A'.Na'}_inv(Kx)}_inv(Kb).
                   {X'.Kx.Ts4'.E4'}_inv(Kt).{B'.Kb.Ts2'.E2'}_inv(Kt))
      =|>
      State':= 10 /\ wrequest(A,B,a_b_IPa,A')
                  /\ wrequest(A,B,a_a_Na,Na')
end role

role intermediate_node1(
  A,B,C,J,X : agent,
  Ka, Kb, Kc, Kj, Kt, Kx : public_key,
  RCV,SND : channel(dy))
played_by B def=
  local
    RDP,REP : protocol_id,
    State : nat,
    Na,Plus,Ts1,E1,Ts2,E2,Ts3,E3,Ts4,E4 : text
  const a_b_IPa, a_b_IPx, a_a_Na : protocol_id
  init State := 1
  transition
    step1.
      State = 1 /\ RCV({RDP.X'.Na'}_inv(Ka).
                   {A'.Ka.Ts1'.E1'}_inv(Kt).{Plus}_inv(Kj))
      =|>
      State':= 6 /\ SND({{RDP.X'.Na'}_inv(Ka)}_inv(Kb).
                     {A'.Ka.Ts1'.E1'}_inv(Kt).{B.Kb.Ts2.E2}_inv(Kt))
                 /\ wrequest(B,A,a_b_IPx,X')
                 /\ wrequest(B,A,a_a_Na,Na')
                 /\ witness(B,C,a_b_IPx,X')
                 /\ witness(B,C,a_a_Na,Na')
    step2.
      State = 6 /\ RCV({{REP.A'.Na'}_inv(Kx)}_inv(Kc).
                   {X'.Kx.Ts4'.E4'}_inv(Kt).{C'.Kc.Ts3'.E3'}_inv(Kt))
      =|>
      State':= 11 /\ SND({{REP.A'.Na'}_inv(Kx)}_inv(Kb).
                     {X'.Kx.Ts4'.E4'}_inv(Kt).{B.Kb.Ts2.E2}_inv(Kt))
                  /\ wrequest(B,C,a_b_IPa,A')
                  /\ wrequest(B,C,a_a_Na,Na')
                  /\ witness(B,A,a_b_IPa,A')
                  /\ witness(B,A,a_a_Na,Na')
end role

role intermediate_node2(
  A,B,C,J,X : agent,
  Ka, Kb, Kc, Kj, Kt, Kx : public_key,
  RCV,SND : channel(dy))
played_by C def=
  local
    RDP,REP : protocol_id,
    State : nat,
    Na,Plus,Ts1,E1,Ts2,E2,Ts3,E3,Ts4,E4 : text
  const a_b_IPa, a_b_IPx, a_a_Na : protocol_id
  init State := 2
  transition
    step1.
      State = 2 /\ RCV({{RDP.X'.Na'}_inv(Ka)}_inv(Kb).
                   {A'.Ka.Ts1'.E1'}_inv(Kt).
                   {B'.Kb.Ts2'.E2'}_inv(Kt).{Plus}_inv(Kj))
      =|>
      State' := 7 /\ SND({{RDP.X'.Na'}_inv(Ka)}_inv(Kc).
                     {A'.Ka.Ts1'.E1'}_inv(Kt).{C.Kc.Ts3.E3}_inv(Kt))
                  /\ wrequest(C,B,a_b_IPx,X')
                  /\ wrequest(C,B,a_a_Na,Na')
                  /\ witness(C,X,a_b_IPx,X')
                  /\ witness(C,X,a_a_Na,Na')
    step2.
      State = 7 /\ RCV({REP.A'.Na'}_inv(Kx).
                   {X'.Kx.Ts4'.E4'}_inv(Kt))
      =|>
      State':= 12 /\ SND({{REP.A'.Na'}_inv(Kx)}_inv(Kc).
                     {X'.Kx.Ts4'.E4'}_inv(Kt).
                     {C.Kc.Ts3.E3}_inv(Kt))
                  /\ wrequest(C,X,a_b_IPa,A')
                  /\ wrequest(C,X,a_a_Na,Na')
                  /\ witness(C,B,a_b_IPa,A')
                  /\ witness(C,B,a_a_Na,Na')
end role

role final_node(
  A,B,C,J,X : agent,
  Ka, Kb, Kc, Kj, Kt, Kx : public_key,
  RCV,SND : channel(dy))
played_by X def=
  local
    RDP,REP : protocol_id,
    State : nat,
    Na,Plus,Ts1,E1,Ts2,E2,Ts3,E3,Ts4,E4 : text
```

```
  const a_b_IPa, a_b_IPx, a_a_Na : protocol_id
  init State := 3
  transition
    step1.
      State = 3 /\ RCV({{RDP.X'.Na'}_inv(Ka)}_inv(Kc).
                   {A'.Ka.Ts1'.E1'}_inv(Kt).
                   {C'.Kc.Ts3'.E3'}_inv(Kt).{Plus}_inv(Kj))
      =|>
      State':= 8 /\ SND({REP.A'.Na'}_inv(Kx).
                    {X.Kx.Ts4.E4}_inv(Kt))
                 /\ wrequest(X,C,a_b_IPx,X')
                 /\ wrequest(X,C,a_a_Na,Na')
                 /\ witness(X,C,a_b_IPa,A')
                 /\ witness(X,C,a_a_Na,Na')
end role

role joker_node(
  A,B,C,J,X : agent,
  Ka, Kb, Kc, Kj, Kt, Kx : public_key,
  RCV,SND : channel(dy))
played_by J def=
  local
    RDP,REP : protocol_id,
    State : nat,
    Na,Plus,Ts1,E1,Ts2,E2,Ts3,E3,Ts4,E4 : text
  const a_b_IPa, a_b_IPx, a_a_Na : protocol_id
  init State := 4
  transition
    step1.
      State = 4 /\ RCV({RDP.X'.Na'}_inv(Ka).
                   {A'.Ka.Ts1'.E1'}_inv(Kt))
      =|>
      State':= 9 /\ SND({RDP.X'.Na'}_inv(Ka).
                    {A'.Ka.Ts1'.E1'}_inv(Kt).{Plus}_inv(Kj))
                 /\ SND({{RDP.X'.Na'}_inv(Ka)}_inv(Kb).
                    {A'.Ka.Ts1'.E1'}_inv(Kt).
                    {B.Kb.Ts2.E2}_inv(Kt).{Plus}_inv(Kj))
                 /\ SND({{RDP.X'.Na'}_inv(Ka)}_inv(Kc).
                    {A'.Ka.Ts1'.E1'}_inv(Kt).
                    {C.Kc.Ts3.E3}_inv(Kt).{Plus}_inv(Kj))
    step2.
      State = 9 /\ RCV({{RDP.X'.Na'}_inv(Ka)}_inv(Kb).
                   {A'.Ka.Ts1'.E1'}_inv(Kt).{B'.Kb.Ts2'.E2'}_inv(Kt))
      =|>
      State':= 14 /\ SND({{RDP.X'.Na'}_inv(Ka)}_inv(Kb).
                     {A'.Ka.Ts1'.E1'}_inv(Kt).
                     {B'.Kb.Ts2'.E2'}_inv(Kt).{Plus}_inv(Kj))
                  /\ SND({{RDP.X'.Na'}_inv(Ka)}_inv(Kc).
                     {A'.Ka.Ts1'.E1'}_inv(Kt).
                     {C.Kc.Ts3.E3}_inv(Kt).{Plus}_inv(Kj))
    step3. State = 14 /\ RCV({{RDP.X'.Na'}_inv(Ka)}_inv(Kc).
                         {A'.Ka.Ts1'.E1'}_inv(Kt).
                         {C'.Kc.Ts3'.E3'}_inv(Kt))
    =|>
    State':= 19 /\ SND({{RDP.X'.Na'}_inv(Ka)}_inv(Kc).
                   {A'.Ka.Ts1'.E1'}_inv(Kt).
                   {C'.Kc.Ts3'.E3'}_inv(Kt).{Plus}_inv(Kj))
end role

role session(
  A,B,C,J,X : agent,
  Ka, Kb, Kc, Kj, Kt, Kx : public_key)
  def=
  local
    RCV1,SND1,RCV2,SND2,RCV3,SND3,RCV4,SND4,RCV5,SND5 : channel(dy)
  composition
    source_node(A,B,C,J,X,Ka,Kb,Kc,Kj,Kt,Kx,RCV1,SND1)
    /\ intermediate_node1(A,B,C,J,X,Ka,Kb,Kc,Kj,Kt,Kx,RCV2,SND2)
    /\ intermediate_node2(A,B,C,J,X,Ka,Kb,Kc,Kj,Kt,Kx,RCV3,SND3)
    /\ final_node(A,B,C,J,X,Ka,Kb,Kc,Kj,Kt,Kx,RCV4,SND4)
    /\ joker_node(A,B,C,J,X,Ka,Kb,Kc,Kj,Kt,Kx,RCV5,SND5)
end role

role environment()
  def=
  const
    a_b_IPa, a_b_IPx, a_a_Na : protocol_id,
    a,b,c,x,j : agent,
    ka,kb,kc,kj,kt,kx : public_key
    intruder_knowledge ={a,b,c,x,ka,kb,kc,kj,kt,kx}
  composition
    session(a,b,c,j,x,ka,kb,kc,kj,kt,kx)
end role

goal
    weak_authentication_on a_b_IPa
    weak_authentication_on a_b_IPx
    weak_authentication_on a_a_Na
end goal

environment()
```

HLPSL SPECIFICATION OF ENDAIRA

```
role source_node(
  A,B,X : agent,
  Ka, Kb, Kx : public_key,
  RCV, SND : channel(dy))
played_by A def=
  local
    Rrep : protocol_id,
    State : nat
  const aa, bb, xx : protocol_id
  init State := 2
  transition
    step1.
      State = 2 /\ RCV({{Rrep.A'.X'.B'}_inv(Kx)}_inv(Kb))
                /\ A' = A /\ B' = B
      =|>
      State':= 5 /\ wrequest(A,B,xx,X') /\ wrequest(A,B,bb,B')
                 /\ wrequest(A,B,aa,A')
end role

role intermediate_node(
  A,B,X : agent,
  Ka, Kb, Kx : public_key,
  RCV,SND : channel(dy))
played_by B def=
  local
    Rrep : protocol_id,
    State : nat
  const aa, bb, xx : protocol_id
  init State := 1
  transition
    step1.
      State = 1 /\ RCV({Rrep.A'.X'.B'}_inv(Kx))
                /\ A' = A /\ B' = B /\ X' = X
      =|>
      State':= 4 /\ SND({{Rrep.A'.X'.B'}_inv(Kx)}_inv(Kb))
                 /\ wrequest(B,X,xx,X') /\ wrequest(B,X,bb,B')
                 /\ wrequest(B,X,aa,A') /\ witness(B,A,xx,X')
                 /\ witness(B,A,bb,B')  /\ witness(B,A,aa,A')
end role

role final_node(
  A,B,X : agent,
  Ka, Kb, Kx : public_key,
  RCV,SND : channel(dy))
played_by X def=
  local
    Rrep : protocol_id,
    State : nat
  const aa, bb, xx : protocol_id
  init State := 0
  transition
    step1.
      State = 0 /\ RCV(start) /\ B' = B /\ X' = X
      =|>
      State':= 3 /\ SND({Rrep.A.X.B}_inv(Kx))
                 /\ witness(X,B,xx,X) /\ witness(X,B,bb,B)
                 /\ witness(X,B,aa,A)
end role

role session(
  A,B,X : agent,
  Ka, Kb, Kx : public_key)
  def=
  local
    RCV1,SND1,RCV2,SND2,RCV3,SND3 : channel(dy)
  composition
     final_node(A,B,X,Ka,Kb,Kx,RCV3,SND3)
    /\ intermediate_node(A,B,X,Ka,Kb,Kx,RCV2,SND2)
    /\ source_node(A,B,X,Ka,Kb,Kx,RCV1,SND1)
end role

role environment()
  def=
  const
    aa, bb, xx : protocol_id,
    a,b,x : agent,
    ka,kb,kx : public_key
    intruder_knowledge ={a,b,x,ka,kb,kx}
  composition
    session(a,b,x,ka,kb,kx)
    /\ session(i,b,x,ka,kb,kx)
    /\ session(a,i,x,ka,kb,kx)
    /\ session(a,b,i,ka,kb,kx)
end role

goal
    weak_authentication_on xx
    weak_authentication_on bb
    weak_authentication_on aa
end goal

environment()
```