Provably Secure On-demand Source Routing in Mobile Ad Hoc Networks[*]

Gergely Ács, Levente Buttyán, and István Vajda
Laboratory of Cryptography and Systems Security (CrySyS)
Department of Telecommunications
Budapest University of Technology and Economics
{acs, buttyan, vajda}@crysys.hu

March 2005

Abstract

Routing is one of the most basic networking functions in mobile ad hoc networks. Hence, an adversary can easily paralyze the operation of the network by attacking the routing protocol. This has been realized by many researchers, and several "secure" routing protocols have been proposed for ad hoc networks. However, the security of those protocols have mainly been analyzed by informal means only. In this paper, we argue that flaws in ad hoc routing protocols can be very subtle, and we advocate a more systematic way of analysis. We propose a mathematical framework in which security can be precisely defined, and routing protocols for mobile ad hoc networks can be analyzed rigorously. Our framework is tailored for on-demand source routing protocols, but the general principles are applicable to other types of protocols too. Our approach is based on the simulation paradigm, which has already been used extensively for the analysis of key establishment protocols, but to the best of our knowledge, it has not been applied in the context of ad hoc routing so far. We also propose a new on-demand source routing protocol, called endairA, and we demonstrate the usage of our framework by proving that it is secure in our model.

Keywords: Mobile ad hoc networks, secure routing, provable security

[*]This technical report is an updated version of our earlier report that appeared on IACR ePrint. In this new version, we extend the adversary model to Active-y-x adversaries and we allow multiple parallel protocol runs. We also slightly modify the endairA protocol, and we propose a few variants of it.

1 Introduction

Routing is one of the most basic networking functions in mobile ad hoc networks. Hence, an adversary can easily paralyze the operation of the network by attacking the routing protocol. This has been realized by many researchers, and several "secure" routing protocols have been proposed for ad hoc networks (see [11] for a survey). However, the security of those protocols have been analyzed either by informal means only, or with formal methods that have never been intended for the analysis of this kind of protocols. In this paper, we present a new attack on Ariadne, a previously published "secure" routing protocol [8]. Other attacks can be found in [4]. These attacks clearly demonstrate that flaws can be very subtle, and therefore, hard to discover by informal reasoning. Hence, we advocate a more systematic approach to analyzing ad hoc routing protocols, which is based on a rigorous mathematical model, in which precise definitions of security can be given, and sound proof techniques can be developed.

Routing has two main functions: route discovery and packet forwarding. The former is concerned with discovering routes between nodes, whereas the latter is about sending data packets through the previously discovered routes. There are different types of ad hoc routing protocols. One can distinguish proactive (e.g., OLSR [5]) and reactive (e.g., AODV [17] and DSR [12]) protocols. Protocols of the latter category are also called on-demand protocols. Another type of classification distinguishes routing table based protocols (e.g., AODV) and source routing protocols (e.g., DSR). In this paper, *we focus on the route discovery part of on-demand source routing protocols*, but we believe that the general principles of our approach are applicable to the route discovery part of other types of protocols too.

At a very informal level, security of a routing protocol means that it can perform its functions even in the presence of an adversary. Obviously, the objective of the adversary is to prevent the correct functioning of the routing protocol. Since we are focusing on the route discovery part of on-demand source routing protocols, in our case, attacks are aiming at achieving that honest nodes receive "incorrect" routes as a result of the route discovery procedure. We will make it more precise later what we mean by an "incorrect" route.

Regarding the capabilities of the adversary, we assume that it can mount active attacks (i.e., it can eavesdrop, modify, delete, insert, and replay messages) from corrupted nodes that have the same communication capabilities as the nodes of the honest participants in the network. This means that the adversary is not all powerful, and it cannot fully control the communication of the honest participants; it can receive only those messages that were transmitted by one of its neighbors, and its transmissions can be heard only by its neighbors. We further assume that the adversary has compromised some identifiers by which we mean that it has compromised the cryptographic keys that are used to authenticate those identifiers. Thus, the adversary can appear as an honest participant under the compromised identities. Using the notation introduced in [8], our adversary is an *Active-y-x* adversary, which means that it controls x corrupted nodes in the network, and it can use y compromised identifiers.

The mathematical framework that we introduce in this paper is based on the so called *simulation paradigm*. This has been successfully used in the analysis of some cryptographic algorithms and some cryptographic protocols (see Section 5 for a very brief overview). However, it has never been applied in the context of ad hoc routing protocols. One of the main contributions of this work is the application of this approach in a new context. Another contribution of this work is the discovery of as yet unknown attacks against previously published ad hoc routing protocols. Finally, yet another contribution is a new on-demand source routing protocol for mobile ad hoc networks, called endairA, which is provably secure in our model, and which may be of independent interest for practitioners.

Preliminary results of this work has been presented in [4]. However, in that paper, we considered only an Active-1-1 adversary, and we did not allow parallel protocol runs. In this paper, we extend our previous results to an Active-y-x adversary, where $x, y \geq 1$, and we allow the simultaneous execution of any number of instances of the route discovery protocol. We also present a new Active-1-2 attack against Ariadne, as well as some extensions and variants of the endairA protocol, which have never been published before.

The rest of the paper is organized as follows: In Section 2, we present a new Active-1-2 attack on Ariadne, and motivate the need for a rigorous analysis technique. In Section 3, we introduce our mathematical framework, which includes a precise definition of security. In Section 4, we present endairA, a new on-demand source routing protocol for ad hoc networks, and we demonstrate the usage of our framework by proving endairA secure. We report on some related work in Section 5, where we also highlight some novelties of our modelling approach with respect to previous applications of the simulation paradigm. Finally, in Section 6, we conclude the paper.

2 An Active-1-2 attack on Ariadne

We have already published attacks against Ariadne and SRP in [4]. In this section, we present a new, as yet unpublished attack against Ariadne. Our goal is to demonstrate that attacks against ad hoc routing protocols can be very subtle, and therefore, difficult to discover. Consequently, it is also difficult to gain sufficient assurances that a protocol is free of flaws. The approach of verifying the protocol for a few number of specific configurations can never be exhaustive, and thus, it is far from being satisfactory as a method for security analysis. The attack presented in this section motivates a more rigorous way of analyzing ad hoc routing protocols, which is the main theme of this paper.

2.1 Operation of Ariadne with MACs

Ariadne has been proposed in [8] 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 [18], on digital signatures, or on MACs. We discuss Ariadne with MACs.

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 computes a MAC on the updated request with a key that it shares with the target. Finally, the MAC is appended to a MAC list in the request, and the request is re-broadcast. The purpose of the per-hop hash value is to prevent removal of identifiers from the accumulated route in the route request.

When the target receives the request, it verifies the per-hop hash by re-computing the initiator's MAC and the per-hop hash value of each intermediate node. Then it verifies the MAC of each intermediate node. 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 obtained from the request, and the MAC of the target on all these elements that is computed with a key shared by the target and the initiator. Each intermediate node passes the reply to the next node on the route (towards the initiator) without any modification. When the initiator receives the reply, it verifies the MAC of the target. If the verification is successful, then it accepts the route returned in the reply.

Although Ariadne does not specify it explicitly, we will nonetheless assume that each node also performs the following verifications when processing route request and route reply messages:

- When a node v receives a route request for the first time, it verifies if the last identifier of the accumulated route in the request corresponds to a neighbor of v. If no identifiers can be found in the accumulated route, then v verifies if the identifier of the initiator corresponds to a neighboring node.
- When a node v receives a route reply, it verifies if its identifier is included in the route carried by the reply. In addition, it also verifies if the preceding identifier (or if there is no preceding identifier, then the identifier of the initiator) and the following identifier (or if there is no following identifier, then the identifier of the target) in the route correspond to neighbors of v.

If these verifications fail, then the message is dropped. Note, however, that the intermediate nodes cannot verify the MACs of the preceding nodes in the route request and the MAC of the target in the route reply, because they do not possess the necessary keys for that.

2.2 The attack

Let us consider now the network configuration illustrated in Figure 1. We assume that the adversary controls two nodes (represented by the black nodes in the figure), but it uses only a single corrupted identifier Z (hence, it is an Active-1-2 adversary). We explain the attack when Ariadne is used with standard MACs, but it also works if TESLA is used, or when digital signatures are used and intermediate nodes do not verify the signature list in the route request (which is an assumption that is compliant with the description of Ariadne in [8]).

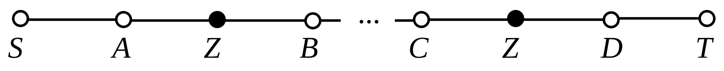

Figure 1: Part of a configuration where an Active-1-2 attack against Ariadne is possible

S initiates a route discovery process toward T. The first adversarial node receives the following route request:

$$msg_1 = (\mathsf{rreq},\ S,\ T,\ id,\ h_A,\ (A), (mac_A))$$

The adversary does not append the MAC of Z to the request, instead, it puts h_A on the MAC list, and re-broadcasts the following request:

$$msg_2 = (\mathsf{rreq},\ S,\ T,\ id,\ h_A,\ (A, Z), (mac_A, h_A))$$

Recall that the intermediate nodes cannot verify the MACs in the request. Note also that MAC functions based on cryptographic hash functions (e.g., HMAC [13]) output a hash value as the MAC, and therefore, h_A looks like a MAC. Hence, B will not detect the attack, and the following request arrives to the second adversarial node:

$$msg_3 = (\mathsf{rreq},\ S,\ T,\ id,\ H(C, \ldots, H(B, h_A)),\ (A, Z, B, \ldots, C), (mac_A, h_A, mac_B, \ldots, mac_C))$$

The adversary removes $B, \ldots, C$ from the node list and the corresponding MACs from the MAC list. The adversary can do this in the following way: By recognizing identifier Z in the accumulated route, the adversary knows that the request passed through the first adversarial node. By looking at the position of identifier Z in the node list, the adversary will know where h_A is on the MAC list. From h_A, the adversary computes $h_Z = H(Z, h_A)$ and a MAC on $(\mathsf{rreq}, S, T, id, h_Z, (A, Z), mac_A)$, and re-broadcasts the following request:

$$msg_4 = (\mathsf{rreq},\ S,\ T,\ id,\ h_Z,\ (A, Z), (mac_A, mac_Z))$$

Since the per-hop hash value and both MACs are correct in msg_4, T will receive a correct request, and returns the following reply:

$$msg_5 = (\mathsf{rrep},\ T,\ S,\ (A, Z, D),\ mac_T)$$

When the reply reaches the second adversarial node, it will forward the following message to C:

$$msg_6 = (\mathsf{rrep},\ T,\ S,\ (A, Z, B, \ldots, C, Z, D),\ mac_T)$$

Note that $B, \ldots, C$ cannot verify the MAC in msg_6. In addition, their identifiers are in the route carried by the reply, and the preceding and following identifiers belong to their neighbors. Therefore, each of them forwards the reply. Finally, when the first adversarial node receives the reply, it removes $B, \ldots, C$ and one of the Z's from the node list:

$$msg_7 = (\mathsf{rrep},\ T,\ S,\ (A, Z, D),\ mac_T)$$

In this way, S receives the route reply that T sent. This means that the MAC verifies correctly and S accepts the route (S, A, Z, D, T), which is non-existent.

It must be noted that in msg_6, the compromised identifier Z appears twice in the node list. Note, however, that Ariadne does not specify that intermediate nodes should check the node list in the reply for repeating identifiers. If each honest node checks only that its own identifier is in the list and that the preceding and following identifiers belong to its neighbors, then the attack works. Moreover, a slightly modified version of the attack would work even if the intermediate nodes checked repeating identifiers in the reply. In that case, the second adversarial node would send the following reply towards S:

$$msg_6' = (\mathsf{rrep},\ T,\ S,\ (A, X, B, \ldots, C, Z, D),\ mac_T)$$

where X can be any identifier that is different from the other identifiers in the node list. With non-negligible probability[1], X is a neighbor of B, and thus, B will pass the reply on, so that the first adversarial node can overhear it. Then, the adversary can remove the identifiers $X, B, \ldots, C$, and send the reply containing the node list (A, Z, D) to A. A will process the reply, because it contains no repeating identifiers and Z is its neighbor. Alternatively, the first adversarial node may send information about the neighborhood of B to the second adversarial node in a proprietary way.

2.3 Some notes on the attack

The attack presented in the previous subsection is very powerful (more powerful than the attack published in [4]), because despite the usage of the per-hop hash mechanism, the adversary manages to

[1]In fact, the probability that X is a neighbor of B is greater than n_B/N, where N is the number of nodes in the network and n_B is the number of B's neighbors.

shorten a discovered route, and therefore, the initiator will probably prefer this short route over others. In other words, the adversary is able to divert the communication between S and T through itself, and then control it.

One can notice that the attack can be prevented by not appending the MACs computed by the intermediate nodes to the route request, but rather updating a single MAC field. More precisely, in this modified version of Ariadne, the route request re-broadcast by the i-th intermediate node F_i would have the following form:

$$(\mathsf{rreq},\ S,\ T,\ id,\ (F_1, \ldots, F_{i-1}, F_i),\ mac_{F_i})$$

where mac_{F_i} is a MAC computed by F_i with the key that it shares with T on the route request that it received:

$$(\mathsf{rreq},\ S,\ T,\ id,\ (F_1, \ldots, F_{i-1}),\ mac_{F_{i-1}})$$

with the convention that $mac_{F_0} = mac_S$.

Besides being more robust than the original version, this modified version of Ariadne has two other advantages. First, there is no need anymore for the per-hop hash mechanism, since the MACs computed by the intermediate nodes can play the same role as the per-hop hash values in the original protocol. Second, route requests are shorter, because they do not contain a per-hop hash value and they contain only a single MAC instead of a MAC list. Note, however, that such an iterative updating would not work with digital signatures, because neither T nor S would be able to re-generate the signatures of the intermediate nodes, which would be necessary for the verification of the route request. When MACs are used, this is not a problem, because each intermediate node uses a key that it shares with T, and thus, T can re-generate the MACs computed by the intermediate nodes, one after the other, which, at the end, makes it possible to verify the last MAC received in the route request.

We note that the authors of Ariadne also come across this modified version of the protocol in [9]. However, they mention it only as an optimization that reduces overhead, and not as a significant modification that increases the security of the protocol.

3 The proposed framework

We follow the so called simulation-based approach to define and prove the security of ad hoc routing protocols. In this approach, two models are constructed for the protocol under investigation: a *real-world model*, which describes the operation of the protocol with all its details in a particular computational model, and an *ideal-world model*, which describes the protocol in an abstract way mainly focusing on the services that the protocol should provide. One can think of the ideal-world model as a description of a specification, and the real-world model as a description of an implementation. Both models contain adversaries. The real-world adversary is an arbitrary process, while the abilities of the ideal-world adversary are usually constrained. The ideal-world adversary models the *tolerable imperfections* of the system; these are attacks that are unavoidable or very costly to defend against, and hence, they should be tolerated instead of being completely eliminated. The protocol is said to be secure if the real-world and the ideal-world models are equivalent, where the equivalence is defined as some form of indistinguishability (e.g., statistical or computational) from the point of view of the honest protocol participants. Technically, security of the protocol is proven by showing that the effects of any real-world adversary on the execution of the real protocol can be *simulated* by an appropriately chosen ideal-world adversary in the ideal-world model.

In the rest of this section, we describe the construction of the real-world model and the ideal-world model, we give a precise definition of security, and briefly discuss some proof techniques, which can be used to prove that a given routing protocol satisfies our definition. We begin the description of the models by introducing two important notions: the notion of a configuration and that of a plausible route.

3.1 Configurations and plausible routes

We model the ad hoc network as an undirected graph $G(V, E)$, where V is the set of vertices, and E is the set of edges. Each vertex represents a node, and there is an edge between two vertices if the corresponding nodes established a wireless link between themselves by successfully running the neighbor discovery protocol. We assume that some of the nodes are under the control of an adversary; we call them corrupted nodes. We assume that corrupted nodes have the same communication capabilities as the non-corrupted nodes in the network. We denote the set of vertices corresponding to corrupted nodes by V^*, and thus, we have $V^* \subset V$.

If two corrupted nodes are neighbors, then they can share information freely (e.g., by sending encrypted messages to each other). In effect, neighboring corrupted nodes can appear as a single node to the other nodes. Hence, without loss of generality, we assume that corrupted nodes are not neighbors in G; if they were, we could merge them into a single corrupted node that would inherit all the neighbors of the original nodes.

Nodes are identified by identifiers in the neighbor discovery protocol and in the routing protocol. We assume that the identifiers are authenticated during neighbor discovery, and therefore, the possibility of a Sybil attack [6] is excluded. We also assume that wormholes [10] are detected at the neighbor discovery level, which means that nodes that are not within each other's radio range are not able to run the neighbor discovery protocol successfully. Hence, the edges in E represent pure radio links.

We assume that the adversary has compromised some identifiers, by which we mean that the adversary has compromised the cryptographic keys that are necessary to authenticate those identifiers. We assume that all the compromised identifiers are distributed to all the corrupted nodes, and they are used in the neighbor discovery protocol and in the routing protocol. On the other hand, we assume that each non-corrupted node uses a single and unique identifier, which is not compromised. We denote the set of all identifiers by L, and the set of the compromised identifiers by L^*.

Let $\mathcal{L} : V \rightarrow 2^L$ be a labelling function, which assigns to each vertex in G a set of identifiers in such a way that for every vertex $v \in V \setminus V^*$, $\mathcal{L}(v)$ is a singleton, and it contains the non-compromised identifier $\ell \in L \setminus L^*$ that is used by the non-corrupted node represented by vertex v; and for every vertex $v \in V^*$, $\mathcal{L}(v)$ contains *all* the compromised identifiers in L^*.

A *configuration* is a triplet $(G(V, E), V^*, \mathcal{L})$. Figure 2 illustrates a configuration, where the solid black vertices are the vertices in V^*, and each vertex is labelled with the set of identifiers that $\mathcal{L}$ assigns to it. Note that the vertices in V^* are not neighboring.

We make the assumption that the configuration is static (at least during the time interval that is considered in the analysis). Thus, we view the route discovery part of the routing protocol as a distributed algorithm that operates on this static configuration. The algorithm is run by the nodes with the aim of finding routes (i.e., sequence of identifiers assigned to the vertices in G), while of course, each node has only a partial knowledge of the configuration.

Intuitively, the minimum that one may require from the route discovery part of the routing protocol is that it returns only existing routes. Our definition of routing security is built on this intuition. Now, we make it more precise what we mean by an existing route.

If there was no adversary, then a sequence $\ell_1, \ell_2, \ldots, \ell_n$ $(n \geq 2)$ of identifiers would be an existing route given that each of the identifiers $\ell_1, \ell_2, \ldots, \ell_n$ are different, and there exists a sequence

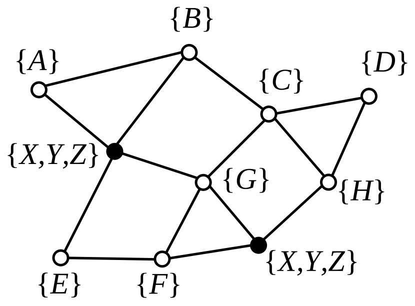

Figure 2: Illustration of a configuration. Corrupted nodes are represented by solid black vertices. Labels on the vertices are identifiers used by the corresponding nodes. Note that corrupted nodes are not neighboring.

$v_1, v_2, \ldots, v_n$ of vertices in V such that $(v_i, v_{i+1}) \in E$ for all $1 \leq i < n$ and $\mathcal{L}(v_i) = \{\ell_i\}$ for all $1 \leq i \leq n$.

However, the situation is more complex due to the adversary that can use the compromised identifiers in L^*. Essentially, we must take into account that the adversary can always emulate the execution of the routing protocol using the compromised identifiers locally within a single corrupted node. Hence, the adversary can always extend any route that passes through a corrupted node with any sequence of compromised identifiers. This is a fact that our definition of security must tolerate, since otherwise we cannot hope that any routing protocol will satisfy it. This observation leads to the following definition:

Definition 1 (Plausible route) *Let $(G(V,E), V^*, \mathcal{L})$ be a configuration. A sequence $\ell_1, \ell_2, \ldots, \ell_n$ of identifiers is a plausible route with respect to $(G(E,V), V^*, \mathcal{L})$ if each of the identifiers $\ell_1, \ell_2, \ldots, \ell_n$ is different, and there exists a sequence $v_1, v_2, \ldots, v_k$ $(2 \leq k \leq n)$ of vertices in V and a sequence $j_1, j_2, \ldots, j_k$ of positive integers such that*

1. $j_1 + j_2 + \ldots + j_k = n$,
2. $(v_i, v_{i+1}) \in E$ $(1 \leq i < k)$,
3. $\{\ell_{J_i+1}, \ell_{J_i+2}, \ldots, \ell_{J_i+j_i}\} \subseteq \mathcal{L}(v_i)$ $(1 \leq i \leq k)$*, where* $J_i = j_1 + j_2 + \ldots + j_{i-1}$ *if* $i > 1$ *and* $J_i = 0$ *if* $i = 1$.

Intuitively, the definition above requires that the sequence $\ell_1, \ell_2, \ldots, \ell_n$ of identifiers can be partitioned into k sub-sequences (condition 1) in such a way that each of the resulting partitions is a subset of the identifiers assigned to a vertex in V (condition 3), and in addition, these vertices form a path in G (condition 2). As an example let us consider again the configuration in Figure 2. It is easy to verify that (A, X, Y, G, C) is a plausible route, because it can be partitioned into $\{A\}, \{X, Y\}, \{G\}, \{C\}$, and these partitions can be assigned to neighboring nodes in the graph. On the other hand, (A, X, G, D, H) is non-plausible, because it can only be partitioned[2] into $\{A\}, \{X\}, \{G\}, \{D\}, \{H\}$, and the partitions $\{G\}$ and $\{D\}$ cannot be assigned to neighboring vertices in the graph.

[2]Note that a non-compromised identifier must always form a separate partition because of the last condition in Definition 1.

3.2 Real-world model

The real-world model that corresponds to a configuration $conf = (G(V, E), V^*, \mathcal{L})$ and adversary $\mathcal{A}$ is denoted by $sys^{\mathsf{real}}_{conf,\mathcal{A}}$, and it is illustrated on the left side of Figure 3. $sys^{\mathsf{real}}_{conf,\mathcal{A}}$ consists of a set $\{M_1, \ldots, M_n, A_1, \ldots, A_m, H, C\}$ of interacting Turing machines, where the interaction is realized via common tapes. Each M_i represents a non-corrupted device that corresponds to a vertex in $V \setminus V^*$, and each A_j represents a corrupted device in V^*. H is an abstraction of higher-layer protocols run by the honest parties, and C models the radio links represented by the edges in E. All machines apart from H are probabilistic.

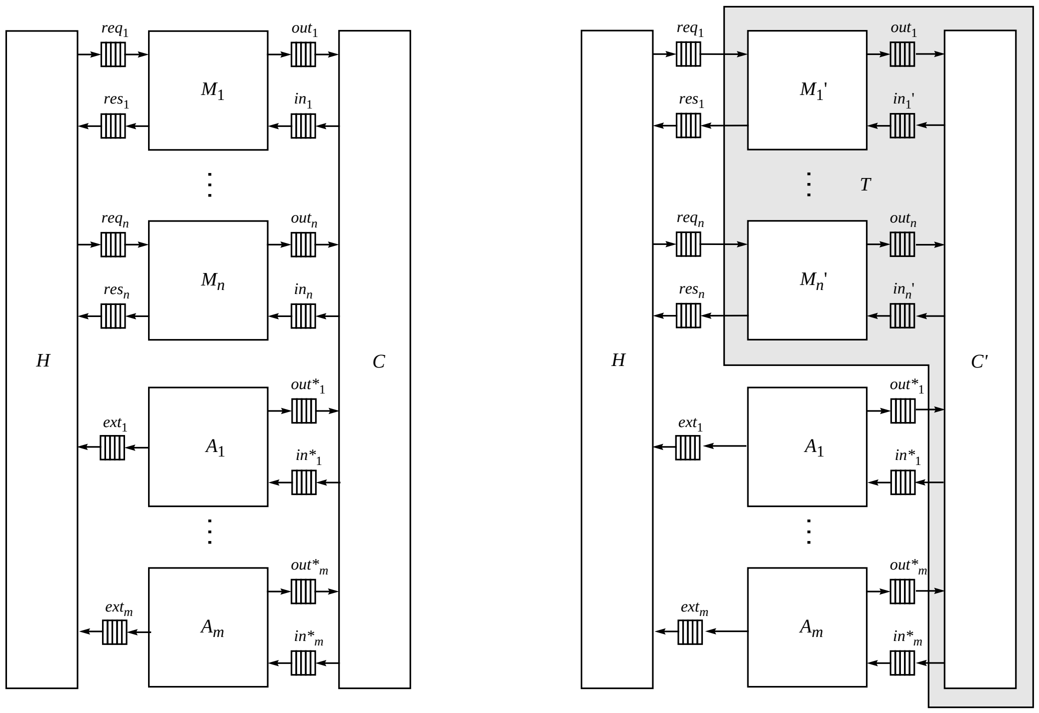

Figure 3: Interconnection of the machines in $sys^{\mathsf{real}}_{conf,\mathcal{A}}$ (on the left side) and in $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ (on the right side)

Each machine is initialized with some input data, which determines its initial state. In addition, the probabilistic machines also receive some random input (the coin flips to be used during the operation). Once the machines have been initialized, the computation begins. The machines operate in a reactive manner, which means that they need to be activated in order to perform some computation. When a machine is activated, it reads the content of its input tapes, processes the received data, updates its internal state, writes some output on its output tapes, and goes back to sleep (i.e., starts to wait for the next activation). Reading a message from an input tape removes the message from the tape, while writing a message on an output tape means that the message is appended to the current content of the tape. Note that each tape is considered as an output tape for one machine and an input tape for another machine. The machines are activated in *rounds* by a hypothetic *scheduler* (not illustrated in Figure 3). In each round, the scheduler activates the machines in the following order: $A_1, \ldots, A_m, H, M_1, \ldots, M_n, C$. In fact, the order of activation is not important, apart from the requirement that C must be activated at the end of the round. Thus, the round ends when C goes back to sleep.

Now, we describe the operation of the machines in more detail:

- *Machine C:* This machine is intended to model the broadcast nature of radio communications. Its task is to read the content of the output tape of each machine M_i and A_j and copy it on the input tapes of *all* the neighboring machines, where the neighbor relationship is determined by the configuration $conf$. Clearly, in order for C to be able to work, it needs to be initialized with some random input, denoted by r_C, and configuration $conf$.

- *Machine H:* This machine models higher-layer protocols (i.e., protocols above the routing protocol) and ultimately the end-users of the non-corrupted devices. H can initiate a route discovery process at any machine M_i by placing a request (c_i, ℓ_{tar}) on tape req_i, where c_i is a sequence number used to distinguish between different requests sent to M_i, and $\ell_{tar} \in L$ is the identifier of the target of the discovery. A response to this request is eventually returned via tape res_i. The response has the form $(c_i, routes)$, where c_i is the sequence number of the corresponding request, and $routes$ is the set of routes found. In some protocols, $routes$ is always a singleton, in others it may contain several routes. If no route found, then $routes = \emptyset$.

  In addition to req_i and res_i, H can access the tapes ext_j. These tapes model an out-of-band channel through which the adversary can instruct the honest parties to initiate route discovery processes. The messages read from ext_j have the form (ℓ_{ini}, ℓ_{tar}), where $\ell_{ini}, \ell_{tar} \in L$ are the identifiers of the initiator and the target, respectively, of the route discovery requested by the adversary. When H reads (ℓ_{ini}, ℓ_{tar}) from ext_j, it places a request (c_i, ℓ_{tar}) in req_i where i is the index of the machine M_i that has identifier ℓ_{ini} assigned to it (see also the description of how the machines M_i are initialized). In order for this to work, H needs to know which identifier is assigned to which machine M_i; it receives this information as an input in the initialization phase.

- *Machine M_i $(1 \leq i \leq n)$:* These machines represent the non-corrupted nodes, which belong to the vertices in $V \setminus V^*$. The operation of M_i is essentially defined by the routing algorithm. M_i communicates with H via its input tape req_i and its output tape res_i. Through these tapes, it receives requests from H for initiating route discoveries and sends the results of the discoveries to H, as described above.

  M_i communicates with the other protocol machines via its output tape out_i and its input tape in_i. Both tapes can contain messages of the form $(sndr, rcvr, msg)$, where $sndr \in L$ is the identifier of the sender, $rcvr \in L \cup \{*\}$ is the identifier of the intended receiver ($*$ meaning a broadcast message), and $msg \in \mathcal{M}$ is the actual protocol message. Here, $\mathcal{M}$ denotes the set of all possible protocol messages, which is determined by the routing protocol under investigation.

  When M_i is activated, it first reads the content of req_i. For each request (c_i, ℓ_{tar}) received from H, it generates a route request msg, updates its internal state according to the routing protocol, and then, it places the message $(\mathcal{L}(M_i), *, msg)$ in out_i, where $\mathcal{L}(M_i)$ denotes the identifier assigned to the node that is represented by M_i.

  When all the requests found in req_i have been processed, M_i reads the content of in_i. For each message $(sndr, rcvr, msg)$ found on in_i, M_i checks if $sndr$ is its neighbor and $rcvr \in \{\mathcal{L}(M_i), *\}$. If these verifications fail, then M_i ignores msg. Otherwise, M_i processes msg and updates its internal state. The way this is done depends on the particular routing protocol in question.

We describe the initialization of M_i after describing the operation of machines A_j.

- *Machine* A_j $(1 \leq j \leq m)$*:* These machines represent the corrupted devices, which belong to the vertices in V^*. Regarding its communication capabilities, A_j is identical to any machine M_i, which means that it can read from in_j^* and write on out_j^* much in the same way as M_i can read from and write on in_i and out_i, respectively. In particular, this means that the A_j cannot eavesdrop messages that were transmitted by devices that are not neighbors of A_j. It also means that "rushing" is not allowed in our model (i.e., A_j must send its messages in a given round before it receives the messages of the same round from other machines). We intend to extend our model and study the effect of "rushing" in our future work.

  While its communication capabilities are similar to that of the non-corrupted devices, A_j may not follow the routing protocol faithfully. In fact, we place no restrictions on the operation of A_j apart from being polynomial-time in the security parameter (e.g., the key size of the cryptographic primitives used in the protocol) and in the size of the network (i.e., the number of nodes). This allows us to consider arbitrary attacks during the analysis. In particular, A_j may delay or delete messages that it would send if it followed the protocol faithfully. In addition, it can modify messages and generate fake ones.

  In addition, A_j may send out-of-band requests to H by writing on ext_j as described above. This gives the power to the adversary to specify who starts a route discovery process and towards which target. Here, we make the restriction that the adversary initiates a route discovery only between non-corrupted nodes, or in other words, for each request (ℓ_{ini}, ℓ_{tar}) that A_j places on ext_j, $\ell_{ini}, \ell_{tar} \in L \setminus L^*$ holds.

  Note that each A_j can write several requests on ext_j, which means that we allow several parallel runs of the routing protocol. On the other hand, we restrict each A_j to write on ext_j only once, at the very beginning of the computation (i.e., before receiving any messages from other machines). This essentially means that we assume that the adversary is *non-adaptive*; it cannot initiate new route discoveries as a function of previously observed messages. We intend to extend our model with adaptive adversaries in our future work.

As it can be seen from the description above, each M_i should know its own assigned identifier, and those of its neighbors in G. M_i receives these identifiers in the initialization phase. Similarly, each A_j receives the identifiers of its neighbors and the set L^* of compromised identifiers.

In addition, the machines may need some cryptographic material (e.g., public and private keys) depending on the routing protocol under investigation. We model the distribution of this material as follows. We assume a function I, which takes only random input r_I, and it produces a vector $I(r_I) = (\kappa_{pub}, \kappa_1, \ldots, \kappa_n, \kappa^*)$. The component κ_{pub} is some public information that becomes known to all A_j and all M_i. κ_i becomes known only to M_i $(1 \leq i \leq n)$, and κ^* becomes known to all A_j $(1 \leq j \leq m)$. Note that the initialization function can model the out-of-band exchange of initial cryptographic material of both asymmetric and symmetric cryptosystems. In the former case, κ_{pub} contains the public keys of all nodes, while κ_i contains the private key that corresponds to the non-compromised identifier $\mathcal{L}(M_i)$, and κ^* contains the private keys corresponding to the compromised identifiers in L^*. In the latter case, κ_{pub} is empty, κ_i contains the symmetric keys known to M_i, and κ^* contains the symmetric keys known to the adversary (i.e., all A_j).

Finally, all M_i and all A_j receive some random input in the initialization phase. The random input of M_i is denoted by r_i, and that of A_j is denoted by r_j^*.

The computation ends when H reaches one of its final states. This happens when H receives a response to each of the requests that it placed on the tapes req_i $(1 \leq i \leq n)$. The output of

$sys^{\mathsf{real}}_{conf,\mathcal{A}}$ is the sets of routes found in these responses. We will denote the output by $Out^{\mathsf{real}}_{conf,\mathcal{A}}(r)$, where $r = (r_I, r_1, \ldots, r_n, r_1^*, \ldots, r_m^*, r_C)$. In addition, $Out^{\mathsf{real}}_{conf,\mathcal{A}}$ will denote the random variable describing $Out^{\mathsf{real}}_{conf,\mathcal{A}}(r)$ when r is chosen uniformly at random.

3.3 Ideal-world model

The ideal-world model that corresponds to a configuration $conf = (G(V, E), V^*, \mathcal{L})$ and adversary $\mathcal{A}$ is denoted by $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$, and it is illustrated on the right side of Figure 3. One can see that the ideal-world model is very similar to the real-world one. Just like in the real-world model, here as well, the machines are interactive Turing machines that operate in a reactive manner, and they are activated by a hypothetic scheduler in rounds. The tapes work in the same way as they do in the real-world model. There is only a small (but important) difference between the operation of M_i' and M_i, and that of C' and C. Below, we will focus on this difference.

Our notion of security is related to the requirement that the routing protocol should return only plausible routes. The differences between the operation of M_i' and M_i, and C' and C, will ensure that this requirement is satisfied in the ideal-world model. In fact, the ideal-world model is meant to be ideal exactly in this sense.

The main idea is the following: Since C' is initialized with $conf$, it can easily identify and mark those route reply messages that contain non-plausible routes. A marked route reply is processed by each machine M_i' in the same way as a non-marked one (i.e., the machines ignore the marker) except for the machine that initiated the route discovery process to which the marked route reply belongs. The initiator first performs all the verifications on the route reply that the routing protocol requires, and if the message passes all these verifications, then it also checks if the message is marked as non-plausible. If so, then it drops the message, otherwise it continues processing (e.g., returns the received route to H). This ensures that in the ideal-world model, every route reply that contains a non-plausible route is caught and filtered out by the initiator of the route discovery[3].

Now, we describe the operation of M_i' and C' in more detail:

- *Machine M_i' $(1 \leq i \leq n)$:* The main difference between M_i' and M_i is that M_i' is prepared to process messages that contain a ***plausibility flag***. The messages that are placed on tape in_i' have the form $(sndr, rcvr, (msg, pf))$, where $sndr$, $rcvr$, and msg are defined in the same way as in the real-world model, and $pf \in \{\mathsf{true}, \mathsf{false}, \mathsf{undef}\}$ is the plausibility flag, which indicates whether msg is a route request ($pf = \mathsf{undef}$), or it is a route reply and it contains only plausible routes ($pf = \mathsf{true}$) or it contains a non-plausible route ($pf = \mathsf{false}$). When machine M_i' reads $(sndr, rcvr, (msg, pf))$ from in_i', it verifies if $sndr$ is its neighbor and $rcvr \in \{\mathcal{L}(M_i'), *\}$. If these verifications are successful, then it performs the verifications required by the routing protocol on msg (e.g., it checks digital signatures, MACs, the route or route segment in msg, etc.). In addition, if msg is a route reply that belongs to a route discovery that was initiated by M_i', then M_i' also checks if $pf = \mathsf{false}$. If so, then M_i' drops msg, otherwise it continues processing it. If msg is not a route reply or M_i' is not the initiator, then pf is ignored. The messages generated by M_i' have no plausibility flag attached to them, and they are placed in out_i.

- *Machine C':* Just like C, C' copies the content of the output tape of each M_i' and A_j onto the

[3]Of course, marked route reply messages can also be dropped earlier during the execution of the protocol for other reasons. What we mean is that if they are not caught earlier, then they are surely removed at latest by the initiator of the route discovery to which they belong.

input tapes of the neighboring machines. However, before copying a message $(sndr, rcvr, msg)$ on any tape in'_i, C' attaches a plausibility flag pf to msg. This is done in the following way:

– if msg is a route request, then C' sets pf to undef;

– if msg is a route reply and all routes carried by msg are plausible with respect to the configuration $conf$, then C' sets pf to true;

– otherwise C' sets pf to false.

Note that C' does not attach plausibility flags to messages that are placed on the tapes in^*_j. Hence, the input and the output tapes of all A_j contain messages of the same format as in the real-world model, which makes it easy to "plug" a real-world adversary into the ideal-world model.

Before the computation begins, each machine is initialized with some input data. This is done in the same way as in the real-world model. The computation ends when H reaches one of its final states. This happens when H receives a response to each of the requests that it placed on the tapes req_i $1 \leq i \leq n$. The output of $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ is the sets of routes returned in these responses. We will denote the output by $Out^{\mathsf{ideal}}_{conf,\mathcal{A}}(r)$, where $r = (r_I, r_1, \ldots, r_n, r^*_1, \ldots, r^*_m, r_C)$. $Out^{\mathsf{ideal}}_{conf,\mathcal{A}}$ will denote the random variable describing $Out^{\mathsf{ideal}}_{conf,\mathcal{A}}(r)$ when r is chosen uniformly at random.

3.4 Definitions of routing security

Now, we are ready to introduce our definition of secure routing:

Definition 2 (Statistical security) *A routing protocol is said to be statistically secure if, for any configuration $conf$ and any real-world adversary $\mathcal{A}$, there exists an ideal-world adversary $\mathcal{A}'$, such that $Out^{\mathsf{real}}_{conf,\mathcal{A}} \overset{\mathsf{s}}{=} Out^{\mathsf{ideal}}_{conf,\mathcal{A}'}$, where $\overset{\mathsf{s}}{=}$ means "statistically indistinguishable".*

Two random variables are statistically indistinguishable if the L_1 distance of their distributions is negligibly small. In fact, it is possible to give a weaker definition of security, where instead of statistical indistinguishability, we require computational indistinguishability. Two random variables are computationally indistinguishable if no feasible algorithm can distinguish their samples (although their distribution may be completely different). Clearly, statistical indistinguishability implies computational indistinguishability, but not vice versa, therefore, computational security is a weaker notion. In this paper, we will only use the concept of statistical security.

Intuitively, statistical security of a routing protocol means that the effect of any real-world adversary in the real-world model can be *simulated* "almost perfectly" by an ideal-world adversary in the ideal-world model. Since, by definition, no ideal-world adversary can achieve that a non-plausible route is accepted in the ideal-world model, it follows that no real-world adversary can exist that can achieve that a non-plausible route is accepted with non-negligible probability in the real-world model, because if such a real-world adversary existed, then no ideal-world adversary could simulate it "almost perfectly". In other words, if a routing protocol is statistically secure, then it can return non-plausible routes only with negligible probability in the real-world model. This negligible probability is related to the fact that the adversary can always forge the cryptographic primitives (e.g., generate a valid digital signature) with a very small probability.

3.5 Proof technique

In order to prove the security of a given routing protocol, one has to find the appropriate ideal-world adversary $\mathcal{A}'$ for any real-world adversary $\mathcal{A}$ such that Definition 2 is satisfied. Due to the constructions of our models, a natural candidate is $\mathcal{A}' = \mathcal{A}$. This is because for any configuration $conf$, the operation of $sys^{\mathsf{real}}_{conf,\mathcal{A}}$ can easily be *simulated* by the operation of $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ assuming that the two systems were initialized with the same random input r. In order to see this, let us assume for a moment that no message is dropped due to its plausibility flag being false in $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$. In this case, $sys^{\mathsf{real}}_{conf,\mathcal{A}}$ and $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ are essentially identical, meaning that in each step the state of the corresponding machines and the content of the corresponding tapes are the same (apart from the plausibility flags attached to the messages in $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$). Since the two systems are identical, $Out^{\mathsf{real}}_{conf,\mathcal{A}}(r) = Out^{\mathsf{ideal}}_{conf,\mathcal{A}}(r)$ holds for every r, and thus, we have $Out^{\mathsf{real}}_{conf,\mathcal{A}} \stackrel{\mathsf{s}}{=} Out^{\mathsf{ideal}}_{conf,\mathcal{A}}$. [4]

However, if some route reply messages are dropped in $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ due to their plausibility flags being set to false, then $sys^{\mathsf{real}}_{conf,\mathcal{A}}$ and $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ may end up in different states and their further steps may not match each other, since those messages are not dropped in $sys^{\mathsf{real}}_{conf,\mathcal{A}}$ (by definition, they have already successfully passed all verifications required by the routing protocol). We call this situation a *simulation failure*. In case of a simulation failure, it might be that $Out^{\mathsf{real}}_{conf,\mathcal{A}}(r) \neq Out^{\mathsf{ideal}}_{conf,\mathcal{A}}(r)$. Nevertheless, the definition of statistical security can still be satisfied, if simulation failures occur only with negligible probability. Hence, when trying to prove statistical security, one tries to prove that for any configuration $conf$ and adversary $\mathcal{A}$, the event of dropping a route reply in $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ due to its plausibility flag being set to false can occur only with negligible probability.

Note that if the above statement cannot be proven, then the protocol can still be secure, because it might be possible to prove the statement for another ideal-world adversary $\mathcal{A}' \neq \mathcal{A}$. In practice, however, failure of a proof in the case of $\mathcal{A}' = \mathcal{A}$ usually indicates a problem with the protocol, and often, one can construct an attack by looking at where the proof failed. Indeed, that is how we discovered an Active-1-1 attack against Ariadne, which is presented in [4].

4 endairA: a provably secure on-demand source routing protocol

Inspired by Ariadne with digital signatures[5], we designed a routing protocol that can be proven to be statistically secure. We call the protocol endairA (which is the reverse of Ariadne), because instead of signing the route request, we propose that intermediate nodes should sign the route reply. In the next subsection, we describe the operation of the basic endairA protocol, and we prove it to be statistically secure. We discuss possible extensions and variants of endairA in Subsection 4.2.

4.1 The basic endairA protocol

The operation and the messages of endairA are illustrated in Figure 4. In endairA, the initiator of the route discovery process generates a route request, which contains the identifiers of the initiator and the target, and a randomly generated request identifier. Each intermediate node that receives the request for the first time appends its identifier to the route accumulated so far in the request, and re-broadcasts the request. When the request arrives to the target, it generates a route reply. The route reply contains the identifiers of the initiator and the target, the accumulated route obtained from the request, and a

[4]In fact, in this case the two random variables have exactly the same distribution.

[5]Ariadne with digital signatures is similar to Ariadne with MACs presented in Section 2 with the difference that instead of computing MACs, the intermediate nodes digitally sign the route request before re-broadcasting it.

digital signature of the target on these elements. The reply is sent back to the initiator on the reverse of the route found in the request. Each intermediate node that receives the reply verifies that its identifier is in the node list carried by the reply, and that the preceding identifier (or that of the initiator if there is no preceding identifier in the node list) and the following identifier (or that of the target if there is no following identifier in the node list) belong to neighboring nodes. Each intermediate node also verifies that the digital signatures in the reply are valid and that they correspond to the following identifiers in the node list and to the target. If these verifications fail, then the reply is dropped. Otherwise, it is signed by the intermediate node, and passed to the next node on the route (towards the initiator). When the initiator receives the route reply, it verifies if the first identifier in the route carried by the reply belongs to a neighbor. If so, then it verifies all the signatures in the reply. If all these verifications are successful, then the initiator accepts the route.

$S \rightarrow *$	:	$(\mathsf{rreq},\ S,\ T,\ id,\ ())$
$A \rightarrow *$	:	$(\mathsf{rreq},\ S,\ T,\ id,\ (A))$
$B \rightarrow *$	:	$(\mathsf{rreq},\ S,\ T,\ id,\ (A,B))$
$T \rightarrow B$	:	$(\mathsf{rrep},\ S,\ T,\ (A,B),\ (sig_T))$
$B \rightarrow A$	:	$(\mathsf{rrep},\ S,\ T,\ (A,B),\ (sig_T, sig_B))$
$A \rightarrow S$	:	$(\mathsf{rrep},\ S,\ T,\ (A,B),\ (sig_T, sig_B, sig_A))$

Figure 4: An example for the operation and messages of endairA. The initiator of the route discovery is S, the target is T, and the intermediate nodes are A and B. id is a randomly generated request identifier. sig_A, sig_B, and sig_T are digital signatures of A, B, and T, respectively. Each signature is computed over the message fields (including the signatures) that precede the signature.

The proof of the following theorem illustrates how the framework introduced in Section 3 can be used in practice.

Theorem 1 *endairA is statistically secure if the signature scheme is secure against chosen message attacks.*

Proof: We provide only a sketch of the proof. We want to show that for any configuration $conf = (G(V,E), V^*, \mathcal{L})$ and any adversary $\mathcal{A}$, a route reply message in $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ is dropped due to its plausibility flag set to false with negligible probability.

In what follows, we will refer to non-corrupted nodes (machines) with their identifiers. Let us suppose that the following route reply is received by a non-corrupted node ℓ_{ini} in $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$:

$$msg = (\mathsf{rrep},\ \ell_{ini},\ \ell_{tar},\ (\ell_1, \ldots, \ell_p),\ (sig_{\ell_{tar}}, sig_{\ell_p}, \ldots, sig_{\ell_1}))$$

Let us suppose that msg passes all the verifications required by endairA at ℓ_{ini}, which means that all signatures in msg are correct, and ℓ_{ini} has a neighbor that uses the identifier ℓ_1. Let us further suppose that msg has been received with a plausibility flag set to false, which means that $(\ell_{ini}, \ell_1, \ldots, \ell_p, \ell_{tar})$ is a non-plausible route in $conf$. Hence, msg is dropped due to the its plausibility flag being false.

Recall that, by definition, corrupted nodes cannot be neighbors. In addition, each non-corrupted node has a single and unique non-compromised identifier. It follows that every route, including $(\ell_{ini}, \ell_1, \ldots, \ell_p, \ell_{tar})$, has a unique *meaningful* partitioning, which is the following: each non-compromised identifier, as well as each sequence of consecutive compromised identifiers should form a partition.

Let $P_1, P_2, \ldots, P_k$ be the unique meaningful partitioning of the route $(\ell_{ini}, \ell_1, \ldots, \ell_p, \ell_{tar})$. The fact that this route is non-plausible implies that at least one of the following two statements holds:

- *Case 1:* There exist two partitions $P_i = \{\ell_j\}$ and $P_{i+1} = \{\ell_{j+1}\}$ such that both ℓ_j and ℓ_{j+1} are non-compromised identifiers, and the corresponding non-corrupted nodes are not neighbors.
- *Case 2:* There exist three partitions $P_i = \{\ell_j\}$, $P_{i+1} = \{\ell_{j+1}, \ldots, \ell_{j+q}\}$, and $P_{i+2} = \{\ell_{j+q+1}\}$ such that ℓ_j and ℓ_{j+q+1} are non-compromised and $\ell_{j+1}, \ldots, \ell_{j+q}$ are compromised identifiers, and the non-corrupted nodes that use ℓ_j and ℓ_{j+q+1} have no common corrupted neighbor.

We show that in both cases, the adversary must have forged the digital signature of a non-corrupted node.

In Case 1, ℓ_{j+1} does not sign the route reply, since it is non-corrupted and it detects that the identifier that precedes its own identifer in the route does not belong to a neighbor. Hence, the adversary must have forged $sig_{\ell_{j+1}}$ in msg.

In Case 2, the situation is more complicated. Let us assume that the adversary has not forged the signature of any of the non-corrupted nodes. ℓ_j must have received

$$msg' = (\mathsf{rrep},\ \ell_{ini},\ \ell_{tar},\ (\ell_1, \ldots, \ell_p),\ (sig_{\ell_{tar}}, sig_{\ell_p}, \ldots, sig_{\ell_{j+1}}))$$

from a corrupted neighbor, say v^*, since ℓ_{j+1} is compromised, and thus, a non-corrupted node would not send out a message with $sig_{\ell_{j+1}}$. In order to generate msg', node v^* must have received

$$msg'' = (\mathsf{rrep},\ \ell_{ini},\ \ell_{tar},\ (\ell_1, \ldots, \ell_p),\ (sig_{\ell_{tar}}, sig_{\ell_p}, \ldots, sig_{\ell_{j+q+1}}))$$

because by assumption, the adversary has not forged the signature of ℓ_{j+q+1}, which is non-compromised. Since v^* has no corrupted neighbor, it could have received msg'' only from a non-corrupted node. However, the only non-corrupted node that would send out msg'' is ℓ_{j+q+1}. This would mean that v^* is a common corrupted neighbor of ℓ_j and ℓ_{j+q+1}, which contradicts the assumption of Case 2. This means that our original assumption cannot be true, and hence, the adversary must have forged the signature of a non-corrupted node.

It should be intuitively clear that if the signature scheme is secure, then the adversary can forge a signature only with negligible probability, and thus, a route reply message in $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ is dropped due to its plausibility flag set to false only with negligible probability. Nevertheless, we sketch how this could be proven formally. The proof is indirect. We assume that there exist a configuration $conf$ and an adversary $\mathcal{A}$ such that a route reply message in $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ is dropped due to its plausibility flag set to false with probability ϵ, and then, based on that, we construct a forger F that can break the signature scheme with probability ϵ/n. If ϵ is non-negligible, then so is ϵ/n, and thus, the existence of F contradicts with the assumption about the security of the signature scheme.

The construction of F is the following. Let puk be an arbitrary public key of the signature scheme. Let us assume that the corresponding private key prk is not known to F, but F has access to a signing oracle that produces signatures on submitted messages using prk. F runs a simulation of $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ where all nodes (machines) are initialized as described in the model, except that the public key of a randomly selected non-corrupted node ℓ_i is replaced with puk. During the simulation, whenever ℓ_i signs a message m, F submits m to the oracle, and replaces the signature of ℓ_i on m with the one produced by the oracle. This signature verifies correctly on other nodes later, since the public verification key of ℓ_i is replaced with puk. By assumption, with probability ϵ, the simulation of $sys^{\mathsf{ideal}}_{conf,\mathcal{A}}$ will result in a route reply message msg such that all signatures in msg are correct and msg contains a non-plausible route. As we saw above, this means that there exists a non-corrupted node ℓ_j such that msg contains the signature sig_{ℓ_j} of ℓ_j, but ℓ_j has never signed (the corresponding part of) msg. Let us assume that $i = j$. In this case, sig_{ℓ_j} is a signature that verifies correctly with the public key puk. Since ℓ_j did not signed (the corresponding part of) msg, F did not call the oracle to generate sig_{ℓ_j}. This means that F managed to produce a signature on a message that verifies correctly with puk. Since F selected ℓ_i randomly, the probability of $i = j$ is $\frac{1}{n}$, and hence, the success probability of F is ϵ/n. □

Besides being provably secure, endairA has another significant advantage over Ariadne (and similar protocols): it is more efficient, because it requires less cryptographic computation overall from the nodes. This is because in endairA, only the processing of the route reply messages involves cryptographic operations, and a route reply message is processed only by those nodes that are in the node list carried in the route reply. In contrast to this, in Ariadne, the route request messages need to be digitally signed by all intermediate nodes; however, due to the way a route request is propagated, this means that each node in the network must sign each and every route request.

4.2 Extensions and variants

Note that in our model presented in Section 3, we made the assumption that the nodes are static (at least during the period of time that is analyzed). The proof of security of endairA relies on this assumption. More precisely, in the proof, we show that if a route is returned by endairA to an honest node, then that route must exist in the graph that represents the network with overwhelming probability. Moreover, once a route has been returned, it remains valid forever, because the graph does not change. This means that under the assumption of static nodes, the basic endairA protocol is not vulnerable to replay attacks. However, if we relax this assumption, and we allow the nodes to move, then the basic protocol has a problem. In that case, when a node initiates a route discovery process and the adversary receives a route request, it can replay an old route reply, and if that reply reaches the initiator, then it will be accepted, despite the fact that it may contain outdated information (i.e., a route that does not exist anymore due to the mobility of the nodes).

Fortunately, we can easily extend the basic endairA protocol to mitigate this problem. All we need to do is to require the target of the route discovery to insert the random request identifier id (received in the route request) in the route reply. Hence, in the extended endairA protocol, the route reply that is passed from intermediate node F_i to node F_{i-1} looks as follows:

$$(\mathsf{rrep},\ S,\ T,\ id,\ (F_1, \ldots, F_n),\ (sig_T, sig_{F_n}, \ldots, sig_{F_i}))$$

Now, when the initiator receives a route reply, it also verifies if it received back the request identifier that it sent in the route request. This makes it practically impossible for the adversary to successfully replay an old route reply that belongs to a previous route discovery process. Of course, when nodes are allowed to move, it is possible that a route reply contains a non-existent route even if there was no attack at all. In order to alleviate this problem, the time interval within which the initiator accepts a reply with a specific request identifier should be appropriately limited.

Another problem with the basic endairA protocol is that it is vulnerable to malicious route request flooding attacks. This is because the route request messages are not authenticated in any way, and hence, an adversary (even without compromising any identity) can initiate route discovery processes in the name of honest nodes. These forged route discovery processes will be carried out completely, including the flooding of the route requests in the whole network, because only the impersonated initiators can detect that they are forged. In order to prevent this, the route request can be digitally signed by the initiator, and rate limiting techniques similar to the one used for Ariadne [8] can be applied with endairA too. Naturally, such extensions put more burden on the nodes, since now they also need to verify the initiator's signature in each route request message and to maintain information that is required by the rate limiting mechanism.

Finally, a practical problem of the basic endairA protocol is that it requires the intermediate nodes to verify every signature in the route reply, and this may turn out to be too "expensive" in some applications. One way to overcome this problem would be to require the intermediate nodes to verify only the signature of the target (this ensures that they all sign the same route that the target signed) and the initiator to verify all the signatures in the route reply (this ensures that the order of the signatures corresponds to the route in the reply). In this case, however, a non-corrupted intermediate node that is on the node list in the reply could be used by two corrupted nodes to pass messages between them, and this may lead to successful attacks.

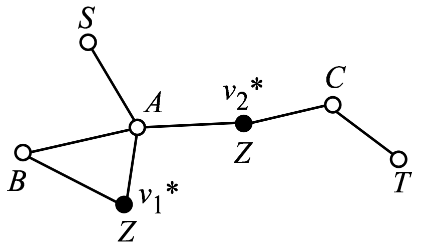

Figure 5: A configuration where an attack against endairA would be possible if the intermediate nodes verified only the signature of the target

In order to illustrate the problem, let us consider the configuration in Figure 5. Let us assume that the initiator is S, the target is T, and the route reply contains the route (A, B, Z, C). This is a non-plausible route, because B and C has no common neighbor that uses identifier Z. Nevertheless, it is possible that S accepts this route. In order to see this, note that the adversarial node v_2^* can send the following message to A in the name of B, after receiving the reply from C:

$$(\mathsf{rrep},\ S,\ T,\ id,\ (A,B,Z,C),\ (sig_T, sig_C, sig_Z))$$

B will accept this message, because the signature of T is valid, A is on the node list, and S and B are neighbors of A. Thus, A will sign the reply and send it to S:

$$(\mathsf{rrep},\ S,\ T,\ id,\ (A,B,Z,C),\ (sig_T, sig_C, sig_Z, sig_A))$$

S will obviously drop this reply, since the signature of B is missing. But the other adversarial node v_1^* will overhear the transmission of A, it can remove the signature of A, and send the following message to B:

$$(\mathsf{rrep},\ S,\ T,\ id,\ (A,B,Z,C),\ (sig_T, sig_C, sig_Z))$$

B will accept this route reply, sign it, and send the following message to A:

$$(\mathsf{rrep},\ S,\ T,\ id,\ (A,B,Z,C),\ (sig_T, sig_C, sig_Z, sig_B))$$

Finally, A accepts this reply again, signs it, and sends the following message to S:

$$(\mathsf{rrep},\ S,\ T,\ id,\ (A,B,Z,C),\ (sig_T, sig_C, sig_Z, sig_B, sig_A))$$

Now, this reply is accepted by S, although it contains a non-plausible route. Note that the above attack is not possible when the original version of endairA is used, because in that case, A would not pass on the reply at the first time, when B's signature is missing from it.

Essentially, the above type of attack is possible because one of the corrupted nodes could pass the reply to the other one through a non-corrupted node that turns out to be on the node list. However, such an attack *always* requires that the victim node processes the reply (at least) twice: once when it is used as a channel between the corrupted nodes, and once when the reply reaches it normally as one of the nodes on the node list. Hence, the optimized version of endairA, where the intermediate nodes verify only the signature of the target, works only if it can be ensured that the intermediate nodes process each route reply only once. They could, for instance, remember the id of each processed reply, and refuse accepting a reply with the same id again. In order to avoid an ever increasing log at the intermediate nodes, a time interval t can be specified by the initiator in the route request. When the request is sent, the initiator would start a timer, and it would accept route reply messages containing id only before the timer reaches t. The intermediate nodes would learn the value of t from the request. They would also start a timer when they process a reply containing id the first time. In addition, they would log id, and drop every further reply containing id. Once their timers reach t, they could remove id from their log, because by that time, the initiator has already timed out, and it would no longer accept any reply with id. Note that this approach does not require any clock synchronization between the nodes.

5 Related work

There are several proposals for secure ad hoc routing protocols (see [11] for a recent overview). However, most of these proposals come with an informal security analysis with all the pitfalls of informal security arguments. In this section, we report on a few exceptions, where some attempts are made to use formal methods for the verification of ad hoc routing protocols.

In [22], the authors try to reach a goal similar to ours but with a different approach. They propose a formal model for ad hoc routing protocols with the aim of representing insider attacks (which correspond to our notion of corrupted nodes). Their model is similar to the strand spaces model [7], which has been developed for the formal verification of key exchange protocols. Routing security is defined in terms of a safety and a liveness property. The liveness property requires that it is possible to discover routes, while the safety property requires that discovered routes do not contain corrupted nodes. In contrast to this, our definition of security allows the protocol to return routes that pass through corrupted nodes, because it seems to be impossible to guarantee that discovered routes do not contain any corrupted node given that corrupted nodes can behave correctly and follow the routing protocol faithfully. Our definition of security corresponds to the informal definitions given in [16] and [8].

Another approach, presented in [15], is based on a formal method, called CPAL-ES, which uses a weakest precondition logic to reason about security protocols. Unfortunately, the work presented in [15] is very much centered around the analysis of SRP [16], and it is not general enough. For instance, the author defines a security goal that is specific to SRP, but no general definition of routing security is given. In addition, the attack discovered by the author on SRP is not a real attack, because it essentially consists in setting up a wormhole between two non-corrupted nodes, and SRP is not supposed to defend against this. In our opinion, wormhole attacks are attacks against the neighbor discovery mechanism and not against routing. On the other hand, the advantage of the approaches of [15] and [22] is that they can be automated.

We must also mention that in [16], SRP has been analyzed by its authors using BAN logic [2]. However, BAN logic has never been intended for the analysis of routing protocols. It has been developed for verifying authentication properties, and there is no easy way to represent the requirements of routing security in it. In addition, BAN logic assumes that the protocol participants are trustworthy [3]. This assumption does not hold in the typical case that we are interested in, namely, when there are corrupted nodes in the network controlled by the adversary that may not follow the routing protocol faithfully. All in all, the BAN analysis of SRP in [16] was inappropriate, which is also confirmed by the fact that even an Active-0-1 adversary can successfully attack it (see [4] for details).

Another set of papers deal with provable security for cryptographic algorithms and protocols (see Parts V and VI of [14] for a survey of the field). However, these papers are not concerned with ad hoc routing protocols. The papers that are the most closely related to the approach we used in this paper are [1], [21], and [19]. These papers apply the simulation paradigm for different security problems: [1] and [21] deal with key exchange protocols, and [19] is concerned with security of reactive systems in general, and secure message transmission in particular. To the best of our knowledge, we are the first who applied the notions of provable security and used the simulation-based approach in the context of routing protocols for wireless ad hoc networks. The main novelties of our model with respect to the models proposed so far for the analysis of cryptographic protocols are the following:

- Our communication model does not abstract away the multi-hop operation of the network. In addition, we model the broadcast nature of radio communications, which allows a node to overhear the transmission of a message that was not intended to him. We also take into account that a radio transmission can usually be received only in a limited range around the sender.

- In contrast to previous models, where the adversary has full control over the communications of the honest nodes, in our model, the adversary can hear only those messages that were transmitted by neighboring nodes, and similarly, the transmissions of the adversary are heard only by its neighbors.

- In our model, it is a hypothetic scheduler, and not the adversary, that schedules the activities of the honest nodes. In addition, this activation is done in rounds. This leads to a sort of synchronous model, where each participant is aware of a global time represented by the current round number. However, *this knowledge has never been exploited in our analysis*. The advantage is that we can retain the simplicity of a synchronous model, without arriving to conclusions that are valid only in synchronous systems.

- The simulation-based approach requires the definition of an ideal-world model, which focuses on *what* the system should do, and it is less concerned about *how* it is done. As a consequence, the ideal-world model usually contains a trusted entity that provides the intended services of the system in a "magical" way. In our model, the role of this trusted entity is played by C', which marks route reply messages that contain non-plausible routes. In addition, we do not limit the capabilities of the ideal-world adversary, but those are the same as the capabilities of a real-world adversary. Consequently, and in contrast to other models, the tolerable imperfections (unavoidable vulnerabilities) of the system are not captured in the capabilities of the ideal-world adversary, but they are embedded in the definition of a plausible route.

6 Conclusion and future work

The main message of this paper is that attacks against ad hoc routing protocols can be subtle and difficult to discover by informal reasoning about the properties of the protocol. We demonstrated this by presenting a novel attack on Ariadne. Another message is that it is possible to adopt rigorous techniques developed for the security analysis of cryptographic algorithms and protocols, and apply them in the context of ad hoc routing protocols in order to gain more assurances about their security. We demonstrated this by proposing a simulation based framework for the security analysis of on-demand source routing protocols. The proposed framework allows us to give a precise definition of security, to model the operation of a given routing protocol in the presence of an adversary, and to prove (or fail to prove) that the protocol is secure. We also proposed a new on-demand source routing protocol, endairA, and we demonstrated the usage of the proposed framework by proving that it is secure in our model. Originally, we developed endairA for purely illustrative purposes, however, it has some noteworthy features that may inspire designers of future protocols.

In this paper, we focused on on-demand source routing protocols. In our future work, we intend to extend our framework for routing table based protocols too (e.g., S-AODV [23] and ARAN [20]).

7 Acknowledgement

The work presented in this paper has partially been supported by the Hungarian Scientific Research Fund (T046664). The second author has been further supported by IKMA and by the Hungarian Ministry of Education (BÖ2003/70).

The authors are thankful to Markus Jakobsson for his comments on an earlier version of this paper.

References

[1] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the ACM Symposium on the Theory of Computing*, 1998.

[2] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[3] M. Burrows, M. Abadi, and R. Needham. Rejoinder to Nessett. *ACM Operating Systems Review*, 24(2):39–40, April 1990.

[4] L. Buttyán and I. Vajda. Towards provable security for ad hoc routing protocols. In *Proceedings of the ACM Workshop on Security in Ad Hoc and Sensor Networks (SASN)*, October 2004.

[5] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). Internet RFC 3626, October 2003.

[6] J. R. Douceur. The Sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[7] J. Guttman. Security goals: packet trajectories and strand spaces. In *Foundations of Security Analysis and Design*, edited by R. Focardi and R. Gorrieri, Springer LNCS 2171, 2000.

[8] Y.-C. Hu, A. Perrig, and D. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the ACM Conference on Mobile Computing and Networking (Mobicom),* 2002.

[9] Y.-C. Hu, A. Perrig, and D. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. to appear in *ACM Mobile Networks and Applications*

[10] Y.-C. Hu, A. Perrig, and D. Johnson. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In *Proceedings of the INFOCOM Conference*, April 2003.

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

[12] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing,* edited by Tomasz Imielinski and Hank Korth, Chapter 5, pages 153–181. Kluwer Academic Publisher, 1996.

[13] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed hashing for message authentication. Internet RFC 2104, February 1997.

[14] W. Mao. *Modern Cryptography: Theory and Practice*. Prentice Hall PTR, 2004.

[15] J. Marshall. An Analysis of the Secure Routing Protocol for mobile ad hoc network route discovery: using intuitive reasoning and formal verification to identify flaws. MSc thesis, Department of Computer Science, Florida State University, April 2003.

[16] P. Papadimitratos and Z. Haas. Secure routing for mobile ad hoc networks. In *Proceedings of SCS Communication Networks and Distributed Systems Modelling Simulation Conference (CNDS),* 2002.

[17] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, February 1999.

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

[19] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2001.

[20] K. Sanzgiri, B. Dahill, B. Levine, C. Shields, and E. Belding-Royer. A secure routing protocol for ad hoc networks. In *Proceedings of the International Conference on Network Protocols (ICNP),* 2002.

[21] V. Shoup. On formal models for secure key exchange (version 4), revision of IBM Research Report RZ 3120, November 1999.

[22] S. Yang and J. Baras. Modeling vulnerabilities of ad hoc routing protocols. In *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks,* October 2003.

[23] M. Zapata and N. Asokan. Securing ad hoc routing protocols. *Proceedings of the ACM Workshop on Wireless Security (WiSe)*, 2002.