Towards Provable Security for Ad Hoc Routing Protocols[*]

Levente Buttyán and István Vajda
Laboratory of Cryptography and System Security (CrySyS)
Department of Telecommunications
Budapest University of Technology and Economics
www.crysys.hu

July 8, 2004

Abstract

We propose a formal framework for the security analysis of on-demand source routing protocols for wireless ad hoc networks. Our approach is based on the well-known simulation paradigm that has been proposed to prove the security of cryptographic protocols. Our main contribution is the application of the simulation approach in the context of ad hoc routing. This involves a precise definition of a real-world model, which describes the real operation of the protocol, and an ideal-world model, which captures what the protocol wants to achieve in terms of security. Both models take into account the peculiarities of wireless communications and ad hoc routing. Then, we give a formal definition of routing security in terms of indistinguishability of the two models from the point of view of honest parties. We demonstrate the usefulness of our approach by analyzing two "secure" ad hoc routing protocols, SRP and Ariadne. This analysis leads to the discovery of as yet unknown attacks against both protocols. Finally, we propose an ad hoc routing protocol that can be proven secure in our model.

1 Introduction

Several "secure" routing protocols have been proposed in the recent past for wireless ad hoc networks [21, 13, 14, 25, 27]. 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 (e.g., SRP was analyzed with BAN logic [7] in [21]). This has at least two annoying consequences:

1. *There is no clear (meaning formal) definition of the term "secure routing"*. Therefore, different authors interpret security in different ways, and design their routing protocols with different requirements in mind. As a consequence, the properties of different proposals are difficult to compare.

2. *There is no mathematically rigorous way to prove a proposed routing protocol secure*. In fact, many of the proposed protocols (e.g., SRP and Ariadne) are flawed in the sense that they do not achieve the properties claimed by their authors; a clear consequence of the lack of a sound proof technique.

[*]Technical Report. Available on-line at *http://eprint.iacr.org/* under report number 2004/159.

The situation described above is somewhat similar to the situation that one could have witnessed in the field of session key establishment protocols in the early 1990's. There, the solution was to come up with definitions and proof techniques on solid mathematical grounds [4, 5, 6, 26, 2]. In this paper, we follow a similar approach, and make the first steps towards a formal model in which one can precisely define what secure routing means and prove (or fail to prove) that a given protocol indeed satisfies that definition (under some cryptographic assumptions).

The organization of the paper is the following: We overview our approach and main contributions in Section 2. We present our model and the formal definition of secure routing in Section 3. We demonstrate the usage and usefulness of our model in Section 4, where we analyze SRP and Ariadne, we describe previously unknown attacks against both protocols, we propose a novel routing protocol, and we prove it secure in our model. In Section 5, we report on related work, and finally, in Section 6, we conclude the paper and give some outlook to the future.

2 Overview of our approach and contributions

Approach. We follow the commonly known simulation approach to prove security of cryptographic protocols [3, 20, 9, 23]. 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 indistinguishability 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.

Contributions. Our main contribution is the application of the approach described above to ad hoc routing protocols. We formally define the real-world and the ideal-world models that capture the basic features of wireless ad hoc networking in general, and ad hoc routing protocols in particular. Most of the models in the literature of provable security for protocols assume the Internet as the underlying networking infrastructure. However, the characteristics of a wireless ad hoc network are very much different from those of the Internet. Hence, the models that have been proposed so far are not directly applicable to wireless ad hoc networks and ad hoc routing protocols. Below, we summarize the peculiarities that we had to deal with when constructing the model; these are the origin of the main differences between our model and the models proposed so far.

Another contribution of this paper is the analysis of two "secure" routing protocols proposed for ad hoc networks: SRP [21] and Ariadne [13]. First, this analysis demonstrates the usage of our approach, and second, it leads to the discovery of as yet unknown attacks against both protocols. This clearly shows the usefulness of our proposal.

Finally, we propose a novel on-demand source routing protocol for wireless ad hoc networks, which can be proven to be secure in our model. This protocol should be viewed as a side effect of our analysis of SRP and Ariadne, and it serves purely illustrative purposes in this paper. However, it has some remarkable features, and we hope that it will inspire protocol designers when building their future protocols.

2.1 Deviations from the standard simulation approach

Now, we overview the main differences between our model and the models proposed so far for the analysis of cryptographic protocols in the context of the simulation approach.

Communication model. One main difference lies in the underlying network model. As we mentioned above, most of the models proposed so far assume that the protocol participants communicate via the Internet (or some similar asynchronous network). Such a network is easily modelled as a single buffer, in which participants place messages, and from which these messages are eventually delivered to their intended recipients. This may be a good model for ad hoc networks if we want to abstract away the multi-hop nature of communications. However, routing protocols are inherently related to the multi-hop operation of the network, and hence, we cannot abstract this away. As a consequence, a single buffer is not an appropriate network model for wireless ad hoc networks.

The peculiarities of wireless networks that we have to deal with include the broadcast nature of radio communications, which allows a party to overhear the transmission of a message that was not destined to him. On the other hand, a radio transmission can be received only in a certain range around the sender. The size of this range mainly depends on the power at which the sender sent the message. For practical reasons, the nodes in an ad hoc network should usually limit their transmission power, which means that messages are received only in a limited neighborhood of their senders. In fact, this is why the communication must be multi-hop in wireless ad hoc networks.

Adversary model. In the models that are based on the Internet assumption, the adversary has the power to control the network buffer. In particular, the adversary can read all messages, it can modify messages before delivering them, and it can delete messages from or place fake messages in the buffer. This is an appropriate model, because in Internet-like networks, having access to some special network elements, such as routers, allows the adversary to have this level of control. On the other hand, in wireless ad hoc networks, an adversary can have a similar level of control over the communications only if it is physically present everywhere. In many applications, this is considered to be very costly, and hence, unrealistic. Therefore, in line with other related papers (e.g., [13]), we assume that the adversary has communication capabilities comparable to those of an average node in the ad hoc network. In our model, the network is represented by a graph, where the vertices are the network nodes (including those controlled by the adversary) and there is an edge between two vertices if the corresponding nodes can hear each other's transmission. Just like any other node, an adversarial node can hear only those messages that were transmitted by a neighboring node in the graph. Similarly, the transmission of an adversarial node is heard only by its neighbors in the graph.

Model of computation. In the models that are based on the Internet assumption, usually the adversary schedules the activities of the honest parties. This is a good model, because many protocols are message driven, meaning that a party becomes active only if it receives some messages. Then, the messages are processed, some output messages are generated, and the party goes back to sleep and starts waiting for new input messages. Hence, by controlling the network and deciding which messages are delivered and when, essentially, the adversary schedules the activities of the honest parties.

On the other hand, in ad hoc networks, the adversary has no full control over the system. This means that some events are beyond his control, and parties can be activated not only by him. Therefore, in our model, the protocol participants (and the adversary) will be activated by a hypothetic scheduler. In addition, this activation is done in *rounds*: in each round, each participant is activated once. This leads to a sort of synchronous model, where each participant is aware of a global time represented by the current round number. One might immediately object that ad hoc networks are not synchronous systems. This is certainly true, and we hasten to note that *knowledge of the current round number is never exploited in our model*. The advantage is that we can retain the simplicity of the presentation, without arriving to conclusions that are valid only in synchronous systems.

The ideal-world model. The simulation 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. This is fully compliant to our expectations with respect to a specification. As a consequence, the ideal-world model usually contains a trusted entity that provides the services of the system in a "magical" way. For instance, the ideal-world model for a session key exchange protocol would contain a trusted host that would generate a random key and return it to the requesting parties only; this model would not deal with the details of the communication between the parties and the details of the cryptographic algorithms used in the real-world system.

Hence, when trying to apply the simulation approach to ad hoc routing protocols, one faces the problem of describing what such protocols should do in an abstract way. However, this seems to be a particularly difficult problem. There are many reasons for this. First of all, ad hoc routing protocols are very complex systems involving many optimizations, which make a precise description of their expected output nearly impossible. Requirements such as the one that the protocol should always return the shortest route between two nodes are simplistic, and in fact, no real protocol satisfies them. As an example, let us consider DSR [16], where a cached route can be returned by an intermediate node as a reply to a route request. Since cashed routes are not based on the most recent topology information, it cannot be guaranteed that DSR returns the current shortest route between the initiator and the target even in the absence of adversaries. Another example is SRP [21], where a rate limiting mechanism is used to regulate the propagation process of the requests. The result is that the propagation of the requests does not solely depend on the topology of the network, but also on the frequency at which the nodes send request messages. In addition, there is always a variation in the processing delays of the nodes, which may lead to returning a suboptimal route just because the propagation of the request was delayed by some node on the optimal route. Since the relationships between the topology changes caused by mobility, the varying processing delays, the amount of traffic generated, and the effects of the optimizations are very complex, the probability distribution of the ideal output of the routing protocol seems to be impossible to determine. Therefore, we adopt the approach of [21] and [13], and we simply require from the ideal routing service that it never returns a non-existent route. While this requirement was stated in [21] and [13] only informally, here, we formalize it.

Another difficulty in the definition of the ideal-world model arises when trying to identify the tolerable imperfections and the capabilities of the ideal-world adversary. It is clear that an adversarial node can always prevent the discovery of some routes, for instance, by deleting route request or route reply messages that it receives. This may be called *explicit* deletion of routes. However, an adversary can delete routes in an *implicit* way too. Many routing protocols use request identifiers that are checked by intermediate nodes to determine if a request of the same route discovery process has already been processed by the node. In order to control the flooding of the network, intermediate nodes usually process a request only if they have not received any request with the same request identifier earlier. This means that if the adversary can send syntactically correct dummy requests with a given request identifier, then he can implicitly delete routes that correspond to those requests that are dropped by intermediate nodes due to the earlier processing of the dummy request. Note that depending on the protocol, the dummy request may be identified only at the target, and thus, it may have a considerable effect on the probability distribution of the set of discovered routes.

Both explicit and implicit deletions must be part of the tolerable imperfections. In other words, the ideal-world adversary must be able to delete those routes that a real-world adversary can delete explicitly or implicitly. However, determining which routes are removable in this way is very difficult, and it is certainly not independent of the details of the routing protocol and the topology of the network.

The problem is further complicated by the observation that the set of route requests (belonging to the same route discovery process) received by the adversary, and thus removable explicitly, depends on which requests the adversary has deleted so far. As an example let us consider part of a topology depicted in Figure 1. A request arrives from node X to the adversarial node A and another node Y. If A does not re-broadcast the request, then eventually it will receive another request belonging to the same discovery process from node Z (i.e., the request that passed through node Y). However, if A does re-broadcast the request received from X, then with high probability, Z will receive this earlier than the request from Y. In this case, Z will not process the request received from Y, and therefore, A will never receive the request that passed through node Y.

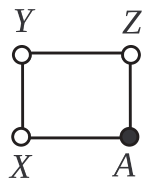

Figure 1: Part of a configuration where the set of route requests received by the adversarial node A depends on which route requests A has deleted so far

In summary, neither the definition of the ideal routing service, nor the definition of the tolerable imperfections seem to be easy to describe in an abstract way, which is independent of the particular protocol under investigation and the configuration (topology and adversarial nodes) of the network. Therefore, we do not define the ideal-world model in terms of an abstract service provider. Instead, the trusted party in our ideal-world model simulates the behavior of the real network, with the difference that it never returns non-existent routes to honest parties (it has a “magical” capability of filtering them out). 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. These are important deviations from the standard simulation approach, but we hope that we have provided sufficient explanations above for the rationale behind them.

3 Model

We consider an ad hoc network of wireless devices. We assume that the radio links between the devices are symmetric, by which we mean that if device v can receive the radio transmission of device

v', then v' can receive the radio transmission of v too. We further assume that each device has a single and unique identifier, which is used, notably, in the neighbor discovery protocol and in the routing protocol. We denote the set of all identifiers by L.

It is convenient to represent an ad hoc network with an undirected labelled graph $G = (V, E, \mathcal{L})$, where V is the set of vertices, E is the set of edges, and $\mathcal{L} : V \rightarrow L$ is a labelling function. Each vertex represents a device, and there is an edge between two vertices v and v', if the corresponding devices can receive each other's radio transmission. Function $\mathcal{L}$ assigns to each vertex the identifier of the corresponding device. Since identifiers are unique, $\mathcal{L}$ must be a bijection.

If $W \subseteq V$, then we will use the shorthand $\mathcal{L}(W)$ to denote the set $\{\mathcal{L}(v) : v \in W\}$ of identifiers assigned to the vertices in W. We introduce a function $\mathcal{N}_G : V \rightarrow 2^V$ that returns the neighboring vertices of a given vertex in G. Formally, $\mathcal{N}_G(v) = \{v' : (v, v') \in E\}$.

Essentially, the routing protocol is a distributed algorithm that operates on G. The algorithm is run by the devices with the aim of finding routes (i.e., sequence of identifiers assigned to the vertices) in G, while of course, each device has only a partial knowledge of G. In some routing protocols, the routes found by the protocol are not returned explicitly, but they are represented implicitly in the state of the devices in form of routing tables. In this paper, we will not be concerned with this kind of protocols. We rather focus on source routing protocols, where the routes are returned explicitly. More specifically, we will be concerned with the route discovery part of on-demand source routing protocols for wireless ad hoc networks. We leave the study of other kinds of ad hoc routing protocols for future work.

We assume an adversary A that wants to subvert the routing service. We assume that A interacts with the system through a single corrupted device. We further assume that A is static, meaning that no further corruption happens during the operation of the system. According to the classification introduced in [13], our adversary is an Active-1-1 attacker. Our adversarial model is deliberately limited, because, as we will see, the analysis is sufficiently instructive and involved even in this case. The more general Active-x-y case and adaptive adversaries are left for future work.

We denote by n the cardinality of V minus 1 (i.e., $|V| = n + 1$). We denote the vertex that represents the corrupted device by $\tilde{v}$, and the vertices that represent the non-corrupted devices by $v_1, v_2, \ldots, v_n$. The pair $(G, \tilde{v})$ is called a *configuration*.

3.1 Real-world model

The real-world model that corresponds to a configuration $conf = (G, \tilde{v})$ and adversary A is denoted by $sys^{\mathsf{real}}_{conf,A}$, and it is illustrated in Figure 2. $sys^{\mathsf{real}}_{conf,A}$ consists of a set $\{M_1, M_2, \ldots, M_n, H, A, C\}$ of Turing machines interacting via *buffers* (or tapes). Each M_i represents the non-corrupted device that corresponds to vertex v_i in G, H is an abstraction of higher-layer protocols run by the honest parties, and A is the adversary, which encompasses the corrupted device corresponding to $\tilde{v}$. Machine C models the radio links represented by the edges of G; it moves messages between the buffers that are connected to it. All machines apart from H are probabilistic.

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 buffers, processes the received data, updates its internal state, writes some output in its output buffers, and goes back to sleep (i.e., starts to wait for the next activation). Reading a message from an input buffer removes the message from the buffer, while writing a message in an output buffer means that the message is appended to the current content

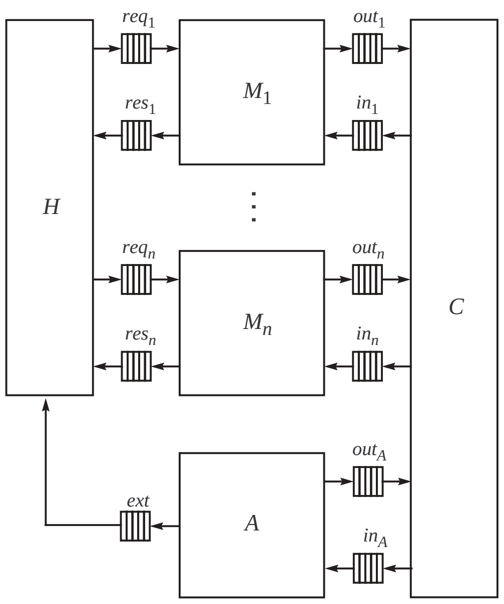

Figure 2: Interconnection of the machines in $sys^{\mathsf{real}}_{conf,A}$

of the buffer. Note that each buffer is considered as an output buffer for one machine and an input buffer for another machine. The machines are activated in *rounds* by a hypothetic *scheduler* (not illustrated in Figure 2). In each round, the scheduler activates the machines in the following order: $H, M_1, \ldots, M_n, A, C$. This means that H is activated first, and then, each machine is activated when the previous machine in the sequence went back to sleep. 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. When activated, it first determines a random order of its input buffers, and then, it processes the content of them in this order[1]. Processing the content of an input buffer out_i $(1 \leq i \leq n)$ consists in reading the content of out_i and copying it in in_j for all j such that $v_j \in \mathcal{N}_G(v_i)$. Similarly, processing the content of out_A means reading the content of out_A and writing it in in_j for all j such that $v_j \in \mathcal{N}_G(\tilde{v})$. Clearly, in order for C to be able to work, it needs to be initialized with some random input, denoted by r_C, and graph G.

- **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}) in buffer 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. c_i is an internal variable of H (i.e., part of its state), which is initialized to 0, and incremented at each time a request is placed in req_i. A response to this request may be returned via buffer 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 returned.

[1]This introduces some non-determinism in the system despite our synchronous model of computation. This non-determinism models the varying processing time of different devices.

In some protocols, $routes$ is always a singleton, in others it is not. If no route found, then $routes = \emptyset$.

In addition to req_i and res_i, H can access buffer ext. This models an out-of-band channel through which the adversary can instruct an honest party to initiate a route discovery process towards a given target. The messages read from ext 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, it first checks if $\ell_{ini} \in \mathcal{L}(\{v_1, \ldots, v_n\})$. If the verification fails, then H ignores the message, otherwise, it places a request (c_i, ℓ_{tar}) in req_i where i is the index of the machine M_i which 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 $(1 \leq i \leq n)$; it receives this information as an input in the initialization phase.

- **Machine M_i:** The operation of M_i is essentially defined by the routing algorithm. M_i communicates with H via its input buffer req_i and its output buffer res_i. Through these buffers, 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 buffer out_i and its input buffer in_i. Both buffers 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.

  In any routing protocol, it must be possible to determine if a protocol message is a route request or a route reply. Hence, there exists a function $type : \mathcal{M} \to \{\mathsf{rreq}, \mathsf{rrep}\}$ that returns the type of any protocol message. In addition, for any protocol message (be it a route request or a route reply), it must also be possible to determine the initiator and the target of the route discovery process to which the message belongs. Therefore, there exist functions $ini : \mathcal{M} \to L$ and $tar : \mathcal{M} \to L$ such that ini returns the identifier of the initiator and tar returns the identifier of the target.

  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 and updates its internal state according to the routing protocol, and then, it places the message $(\mathcal{L}(v_i), *, msg)$ in out_i.

  Once all the requests found in req_i have been processed, M_i reads the content of in_i. The messages found in in_i are processed in the order as they are found in in_i. For each message $(sndr, rcvr, msg)$, M_i checks if $sndr \in \mathcal{L}(\mathcal{N}_G(v_i))$ and $rcvr \in \{\mathcal{L}(v_i), *\}$. If these verifications fail, then M_i ignores msg. Otherwise, M_i processes msg and updates its internal state according to the routing protocol. We distinguish the following two cases:

  $type(msg) = \mathsf{rreq}$: If $tar(msg) \neq \mathcal{L}(v_i)$ (i.e., M_i is not the target of the route discovery), then depending on the content of msg and the current state of M_i, msg may be dropped and no output is generated by M_i. Alternatively, M_i may re-broadcast msg. In this case, M_i generates the appropriate message msg' (e.g., appends its identifier to the route accumulated so far in the request) and places $(\mathcal{L}(v_i), *, msg')$ in out_i.

  If $tar(msg) = \mathcal{L}(v_i)$ (i.e., M_i is the target of the route discovery), then M_i may generate a route reply msg' according to the routing protocol, and place the message $(\mathcal{L}(v_i), \ell, msg')$ in out_i, where, $\ell \in \mathcal{L}(\mathcal{N}_G(v_i))$ is the identifier of the first device on the route that the route reply should follow. In many protocols, each route reply message contains a single route, and the route reply should follow the reverse of that. In this case, ℓ can be unambiguously determined from the route carried by the route reply. Alternatively, if the route reply contains multiple routes or it should not follow the reverse route, then the routing protocol specifies how the target should determine ℓ unambiguously from all the corresponding route requests it received.

$type(msg) = \mathsf{rrep}$: If $ini(msg) \neq \mathcal{L}(v_i)$ (i.e., M_i is not the initiator of the route discovery to which msg belongs), then depending on the content of msg and the current state of M_i, msg may be dropped and no output is generated by M_i. Alternatively, M_i may forward msg, in which case, M_i generates the appropriate protocol message[2] msg', and places the message $(\mathcal{L}(v_i), \ell, msg')$ in out_i, where $\ell \in \mathcal{L}(\mathcal{N}_G(v_i))$ is the identifier of the next device on the route that the route reply should follow. As described above, the routing protocol should specify for M_i how to determine ℓ. If $ini(msg) = \mathcal{L}(v_i)$ (i.e., M_i is the initiator of the route discovery), then M_i may generate a response to H[3] and send it via res_i.

We describe the initialization of M_i after describing the adversary's machine A.

- **Machine A:** This machine represents the adversary. Regarding its communication capabilities, A is identical to any machine M_i, which means that it can read from in_A and write in out_A much in the same way as M_i can read from and write in in_i and out_i, respectively. However, A may not operate according to the routing protocol. In fact, we place no restrictions on the operation of A apart from being polynomial-time in the security parameter k and in the size of the network n. This allows us to consider arbitrary attacks during the analysis. In particular, A may delay or delete messages that it would send if it followed the protocol faithfully. In addition, it can arbitrarily modify messages and generate fake ones.

  In addition, A may send an out-of-band request to H by writing in ext as described above. This gives the power to A to specify who starts a route discovery process and towards which target. However, in order to simplify the analysis, we restrict A to send only a single request via ext. This essentially means that a single route discovery process will take place in our model, or in other words, we do not consider parallel runs of the protocol. It is important to emphasize that this restriction is made only to simplify the analysis in this paper; the model itself is sufficiently rich to capture parallel protocol runs by allowing A to send multiple requests via ext. This feature will be exploited in future papers.

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. Hence, M_i receives $\mathcal{L}(v_i)$ and $\mathcal{L}(\mathcal{N}_G(v_i))$ in the initialization phase. Similarly, A receives $\mathcal{L}(\tilde{v})$ and $\mathcal{L}(\mathcal{N}_G(\tilde{v}))$.

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 $I(r_I) = (\kappa_{pub}, \kappa_1, \ldots, \kappa_n, \tilde{\kappa})$. The component κ_{pub} is some public information that becomes known to A and all M_i's. For $1 \leq i \leq n$, κ_i becomes known only to M_i, and $\tilde{\kappa}$ becomes known only to A. 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 devices, while κ_i $(1 \leq i \leq n)$ contains the private key of the non-corrupted device

[2]Often $msg' = msg$, but in general, it may not be the case.
[3]M_i may wait for a number of route replies before returning a response.

corresponding to v_i, and $\tilde{\kappa}$ contains the private key of the corrupted device corresponding to $\tilde{v}$. In the latter case, κ_{pub} is empty, and κ_i and $\tilde{\kappa}$ contain the symmetric keys known to the non-corrupted device corresponding to v_i and the corrupted device corresponding to $\tilde{v}$, respectively.

Finally, all M_i and A receive some random input in the initialization phase. The random input of M_i is denoted by r_i, and that of A is denoted by r_A.

The computation ends when H reaches one of its final states. In our simplified case (i.e., when A is restricted to send a single out-of-band request to H), this happens when H reads a response from one of its input buffers res_i that corresponds to the single request it placed in req_i. The output of $sys^{\mathsf{real}}_{conf,A}$ is the set of routes found in this response. We will denote the output by $view^{\mathsf{real}}_{conf,A}(r)$, where $r = (r_I, r_1, \ldots, r_n, r_A, r_C)$. In addition, $view^{\mathsf{real}}_{conf,A}$ will denote the random variable describing $view^{\mathsf{real}}_{conf,A}(r)$ when r is uniformly chosen.

3.2 Ideal-world model

The ideal-world model that corresponds to a configuration $conf = (G, \tilde{v})$ and adversary A is denoted by $sys^{\mathsf{ideal}}_{conf,A}$, and it is illustrated in Figure 3. $sys^{\mathsf{ideal}}_{conf,A}$ consists of a set $\{H, T, A\}$ of interacting Turing machines too, where H is the same as in the real-world model, T is intended to model the ideal operation of the routing protocol, and A is the ideal-world adversary. T and A are probabilistic.

As we mentioned earlier in Section 2, the ideal routing service should never return non-existent routes. Hence the role of T will be to emulate the behavior of the real network, and to ensure that route reply messages that contain non-existent routes are identified and filtered out. This is achieved in the following way: The internal structure of T (indicated in gray in Figure 3) is identical to the structure of the real-world model (i.e., T runs machines M'_i and C', which work essentially in the same way as M_i and C do in the real-world model). This ensures that T can emulate the operation of the real network. On the other hand, since C' is initialized with G, it can easily identify and mark as corrupted those route reply messages that contain routes that do not exist in G. A corrupted route reply is processed by each machine M'_i in the same way as a non-corrupted one (i.e., the machines ignore the corruption flag) except for the machine that initiated the route discovery process to which the corrupted 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 corrupted. If so, then it drops the message, otherwise it continues processing (e.g., returns the received route to H). This means that in the ideal-world model, every route reply that contains a non-existent route is catched and filtered out by the initiator of the route discovery[4].

Since T emulates the operation of the real-world model, the attacks that we allow against T should also be the same as those in the real-world model. Therefore, in our approach, the capabilities of an ideal-world adversary will be identical to that of a real-world adversary. This is why we denote both adversaries by A.

Just like in the real-world model, here as well, the machines operate in a reactive manner. They are activated by a hypothetic scheduler in rounds, and in the following order in each round: H, T, A, T. Note that T is activated twice in each round. The buffers work in the same way as they do in the real-world model.

The operation of H and A is the same as in the real-world model. Now, we describe the operation of T in more details:

[4]Of course, corrupted 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.

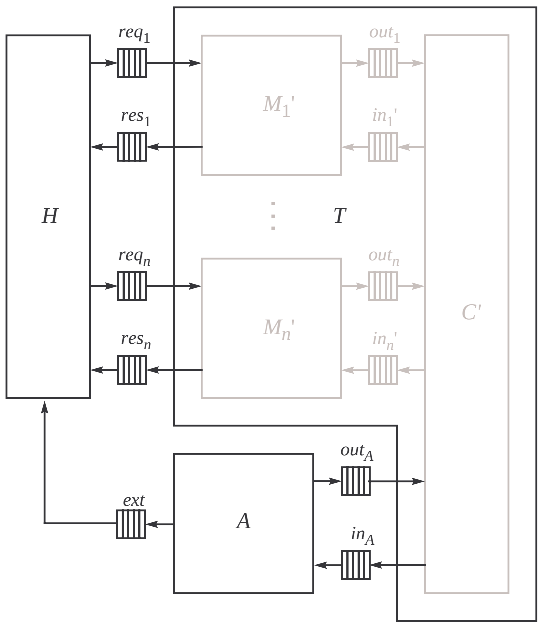

Figure 3: Interconnection of the machines in $sys^{\mathsf{ideal}}_{conf,A}$

- **Machine T:** T runs a set $\{M_1', M_2', \ldots, M_n', C'\}$ of machines, where M_i' and C' are essentially the same as M_i and C, respectively, in the real-world model. The difference between M_i and M_i' is that M_i' is prepared to process messages that contain a corruption flag. The difference between C and C' is that C' attaches a corruption flag to messages that it outputs.

  In each round, when activated the first time, T activates machines $M_1', M_2', \ldots, M_n'$ in this order. Then it goes back to sleep and waits to be activated the second time. When activated the second time, T activates machine C'. When C' finishes its task, T goes back to sleep (and the round ends).

  The messages that are placed in buffer in_i' $(1 \leq i \leq n)$ by C' have the form $(sndr, rcvr, (msg, cf))$, where $sndr$, $rcvr$, and msg are defined in the same way as in the real-world model, and $cf \in \{\top, \bot\}$ is a *corruption flag*, which indicates whether msg is corrupted ($\top$) or not ($\bot$). The messages that are placed in buffers out_i $(1 \leq i \leq n)$, out_A, and in_A have the same form as in the real-world model (i.e., they have no corruption flag attached). Note that the input and the output buffers of A contain messages of the same format as in the real-world model, and therefore, a real-world adversary can easily be "plugged in" the ideal-world model.

  When machine M_i' reads $(sndr, rcvr, (msg, cf))$ from in_i', it verifies if $sndr \in \mathcal{L}(\mathcal{N}_G(v_i))$ and $rcvr \in \{\mathcal{L}(v_i), *\}$. 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 $type(msg) = \mathsf{rrep}$ and $ini(msg) = \mathcal{L}(v_i)$, then M_i' checks if $cf = \top$. If so, then M_i' drops msg, otherwise it continues processing it. If $type(msg) \neq \mathsf{rrep}$ or $ini(msg) \neq \mathcal{L}(v_i)$, then cf is not checked. The messages generated by M_i' have no corruption flags attached to them, and they are placed in out_i.

  Just like C, C' copies the content of the output buffer of each M_i' (and A) into the input buffers of the neighbors of M'_i (and A). However, before copying a message $(sndr, rcvr, msg)$ in any buffer in'_i, C' must attach a corruption flag cf to msg. This is done in the following way:

- if $type(msg) = \mathsf{rreq}$, then C' sets cf to $\perp$;
- if $type(msg) = \mathsf{rrep}$ and all routes carried by msg are existing routes in G, then C' sets cf to $\perp$;
- otherwise C' sets cf to $\top$.

C' does not attach corruption flags to messages that are placed in in_A.

Before the computation begins, each machine is initialized with some input data. H and A receives the same initial input as in the real-world model. The initialization of T consists in the initialization of all M'_i and C'. Every M'_i and C' receive the same initial input as M_i and C, respectively, in the real-world model.

The computation ends when H reaches one of its final states. Since A is restricted to send a single out-of-band request to H, this happens when H reads a response from one of its input buffers res_i that corresponds to the single request it placed in req_i. The output of $sys^{\mathsf{ideal}}_{conf,A}$ is the set of routes found in this response. We will denote the output by $view^{\mathsf{ideal}}_{conf,A}(r')$, where $r' = (r'_I, r'_1, \dots, r'_n, r'_A, r'_C)$. $view^{\mathsf{ideal}}_{conf,A}$ will denote the random variable describing $view^{\mathsf{ideal}}_{conf,A}(r')$ when r' is uniformly chosen.

3.3 Definition of secure routing

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

Definition 1. *A routing protocol is said to be (computationally)* ***secure*** *if, for any configuration $conf$ and any real-world adversary A, there exists an ideal-world adversary A', such that $view^{\mathsf{real}}_{conf,A} \overset{\mathsf{c}}{=} view^{\mathsf{ideal}}_{conf,A'}$, where $\overset{\mathsf{c}}{=}$ means "computationally indistinguishable".*

In fact, Definition 1 describes the standard requirement we have on protocols in terms of security. However, some protocols may satisfy the following stronger definitions:

Definition 2. *A routing protocol is said to be* ***statistically secure*** *if the same holds as in Definition 1 but with $\overset{\mathsf{s}}{=}$ instead of $\overset{\mathsf{c}}{=}$, where $\overset{\mathsf{s}}{=}$ means "statistically indistinguishable".*

Definition 3. *A routing protocol is said to be* ***perfectly secure*** *if the same holds as in Definition 1 but with $\overset{\mathsf{d}}{=}$ instead of $\overset{\mathsf{c}}{=}$, where $\overset{\mathsf{d}}{=}$ means "equally distributed".*

The meaning of $\overset{\mathsf{d}}{=}$ should be clear. Two random variables are statistically indistinguishable if the L_1 distance of their distributions is negligibly small. Two random variables are computationally indistinguishable if no feasible algorithm can distinguish their samples (although their distribution may be completely different). Clearly, $\overset{\mathsf{d}}{=}$ implies $\overset{\mathsf{s}}{=}$, which implies $\overset{\mathsf{c}}{=}$.

Intuitively, perfect security of a protocol means that everything that a real-world adversary can achieve against the protocol in the real-world model, an ideal-world adversary can also achieve in the ideal-world model. Since in the ideal-world model, the ideal-world adversary cannot achieve that a non-existent route is returned to H, it follows that for perfectly secure protocols, H cannot receive a non-existent route in the real-world model. For statistically secure protocols the same is true with overwhelming probability. For (computationally) secure protocols, the view of the honest parties in the real-world model cannot be efficiently distinguished from their view in the ideal-world model, and therefore, as far as any practical application is concerned, the real-world model is equivalent to the ideal-world model (where non-existent routes are never returned).

3.4 Proof technique

In order to prove the security of a given routing protocol, one has to find the appropriate ideal-world adversary A' for any real-world adversary A such that at least Definition 1 is satisfied. In our model, a natural candidate is $A' = A$. This is because for any configuration $conf$, the operation of $sys^{\mathsf{real}}_{conf,A}$ can easily be *simulated* by the operation of $sys^{\mathsf{ideal}}_{conf,A}$ 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 corruption flag being set in $sys^{\mathsf{ideal}}_{conf,A}$. In this case, $sys^{\mathsf{real}}_{conf,A}$ and $sys^{\mathsf{ideal}}_{conf,A}$ are essentially identical, meaning that in each step the state of the corresponding machines and the content of the corresponding buffers are the same (apart from the the corruption flags attached to the messages in $sys^{\mathsf{ideal}}_{conf,A}$). Since the two systems are identical, $view^{\mathsf{real}}_{conf,A}(r) = view^{\mathsf{ideal}}_{conf,A}(r)$ holds for every r, and thus, we have perfect security: $view^{\mathsf{real}}_{conf,A} \overset{\mathsf{d}}{=} view^{\mathsf{ideal}}_{conf,A}$.

However, it is possible that some route reply messages are dropped in $sys^{\mathsf{ideal}}_{conf,A}$ due to their corruption flags being set to $\top$. In this case, since those messages are not dropped in $sys^{\mathsf{real}}_{conf,A}$ (by definition, they have already successfully passed all verifications required by the routing protocol), $sys^{\mathsf{real}}_{conf,A}$ and $sys^{\mathsf{ideal}}_{conf,A}$ may end up in different states and their further steps may not match each other. We call this situation a *simulation failure*. In case of a simulation failure, it might be that $view^{\mathsf{real}}_{conf,A}(r) \neq view^{\mathsf{ideal}}_{conf,A}(r)$. 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 A, the event of dropping a route reply in $sys^{\mathsf{ideal}}_{conf,A}$ due to its corruption flag being set to $\top$ can occur only with negligible probability.

Finally, (computational) security can usually be proven in an indirect manner. For this, it is first assumed that $view^{\mathsf{real}}_{conf,A}$ and $view^{\mathsf{ideal}}_{conf,A}$ can be distinguished by an efficient algorithm $\mathcal{D}$, and then, a forger is constructed that uses $\mathcal{D}$ to break the underlying cryptographic primitive (e.g., a digital signature scheme) of the protocol.

4 Usage of the model

In this section, we show how the model can be used for analyzing routing protocols. In particular, we analyze the route discovery part of SRP (Secure Routing Protocol) and Ariadne with signatures. This analysis leads to as yet unknown attacks against both protocols. Then, we propose a protocol, which we prove to be statistically secure in our model. Throughout this section, we provide only sketches of proofs for several claims. These sketches can be made rigorous proofs in our model, but we omit this for making the presentation easier to follow.

4.1 Analysis of SRP

Operation of SRP. SRP has been proposed in [21] as an extension header for on-demand source routing protocols such as DSR [16] and the Interzone Routing Protocol of ZRP [12]. In what follows, we assume that SRP is a stand-alone protocol with basic features similar to that of DSR. This makes the presentation simpler, and at the same time, it does not weakens our results.

The operation of SRP and the format of SRP messages are illustrated in Figure 4. The initiator of the route discovery generates a route request message and broadcasts it to its neighbors. The integrity of this route request is protected by a MAC that is computed with a key shared by the initiator and the target of the discovery. Each intermediate node that receives the route request for the first time

S $\rightarrow *$	:	(rreq, S, D, id, sn, mac_{S}, ())
B $\rightarrow *$	:	(rreq, S, D, id, sn, mac_{S}, (B))
C $\rightarrow *$	:	(rreq, S, D, id, sn, mac_{S}, (B, C))
D $\rightarrow$ C	:	(rrep, S, D, id, sn, (B, C), mac_{D})
C $\rightarrow$ B	:	(rrep, S, D, id, sn, (B, C), mac_{D})
B $\rightarrow$ S	:	(rrep, S, D, id, sn, (B, C), mac_{D})

Figure 4: Operation example of SRP and format of SRP messages. The identifier of the initiator of the route discovery is S, the identifier of the target is D, and the identifiers of the intermediate nodes are B and C. id is a randomly generated query identifier, sn is a query sequence number maintained by S and D, mac_{S} is the MAC generated by S that covers the fields rreq, S, D, id, and sn, and mac_{D} is the MAC generated by D that covers the fields rrep, S, D, id, sn, and (B, C).

appends its identifier to the request and re-broadcasts it. The MAC in the request is not checked by the intermediate nodes (as they do not know the key with which it was computed), and they do not append their own MACs either. When the route request reaches the target of the route discovery, it contains the list of identifiers of the intermediate nodes that passed the request on. This list is considered as a route found between the initiator and the target.

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

The target may receive several route requests that belong to the same route discovery process[5], and it sends a reply to each of these requests. It is assumed that the initiator waits for some time (possibly defined by a timeout parameter), and then it outputs the set of routes collected from all the replies it received.

Although SRP does not specify it (as it should be part of the base protocol to which SRP is added as an extension), we will nonetheless assume that each node also performs the following verifications when processing SRP messages:

- If a node v receives a route request for the first time, then it verifies if the last identifier of the accumulated route in the request corresponds to a neighbor of v. If the accumulated route does not contain any identifiers, then v verifies if the identifier of the initiator corresponds to a neighboring node. If these verifications fail, then the request is dropped.

- If an intermediate node v receives a route reply, then it verifies if its identifier is included in the route carried by the reply. In addition, it also verifies if the identifier that precedes and the identifier that follows v's identifier in the route correspond to neighboring nodes. If there is no preceding identifier, then v verifies if the identifier of the initiator corresponds to a neighbor. If there is no following identifier, then v verifies if the identifier of the target corresponds to a neighbor. If these verifications fail, then the reply is dropped.

[5]Since the neighbors of the target re-broadcast the request at most once, the target can receive at most as many requests as the number of its neighbors.

- When the initiator receives a route reply, it verifies if the first identifier in the route carried by the reply corresponds to a neighboring node. If this verification fails, then the reply is dropped.

These verifications are quite simple, yet make the protocol more resistant against attacks by identifying non-existent routes in the protocol messages as early as possible.

Analysis. SRP is not secure in our model. Below we present some observations that one could make when attempting to prove the security of SRP. These observations actually lead to the discovery of an attack against SRP.

In the following discussion, we will refer to the machines that represent the devices in the network by their labels. This does not lead to ambiguity since the labelling function $\mathcal{L}$ is a bijection.

Let us suppose that for some configuration $conf = (G, \tilde{v})$ and adversary A, the following message is received by a non-corrupted machine ℓ_{ini} in $sys^{\mathsf{ideal}}_{conf,A}$:

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

Let us further suppose that msg has been received with a corruption flag set to $\top$, and that msg passed all the verifications required by SRP at ℓ_{ini}. This means that $mac_{\ell_{tar}}$ is correct, ℓ_1 is a neighbor of ℓ_{ini}, and $(\ell_{ini}, \ell_1, \ldots, \ell_p, \ell_{tar})$ is a non-existent route in G.

Observation 1: Given that the assumptions above hold, adversary A must have output msg.

Sketch of the proof: Let us assume that A has never output msg. This means that only non-corrupted machines have output it. In other words, ℓ_{ini} received msg from a non-corrupted machine, who received it from another non-corrupted machine, etc. Note that a non-corrupted machine ℓ processes msg only if it was sent to it (i.e., a non-corrupted machine does not process overheard messages). Furthermore, ℓ passes on msg only if it finds itself in the list $(\ell_1, \ldots, \ell_p)$ and if the preceding machine on the list is a neighbor of ℓ. All these observations lead to the conclusion that msg must have reached ℓ_{ini} by passing through $\ell_p, \ldots, \ell_1$. This contradicts with the assumption that $(\ell_{ini}, \ell_1, \ldots, \ell_p, \ell_{tar})$ is a non-existent route. ■

Observation 2: Given that the assumptions above hold, machine ℓ_{tar} has output msg with overwhelming probability.

Sketch of the proof: We know from Observation 1 that A has output msg. Let the earliest round in which this happened be ρ. Since $mac_{\ell_{tar}}$ in msg is a correct MAC, A can generate msg by himself only with negligible probability. So, with overwhelming probability, A received msg in round $\rho' \leq \rho$. Since correct machines apart from ℓ_{tar} output msg only if they received it earlier, there must be a round $\rho'' < \rho'$ in which ℓ_{tar} generated and output msg. ■

By assumption, ℓ_1 is a neighbor of ℓ_{ini}, and $(\ell_{ini}, \ell_1, \ldots, \ell_p, \ell_{tar})$ is a non-existent route. This means that there exists $1 \leq i \leq p$ such that ℓ_i and ℓ_{i+1} (where ℓ_{p+1} stands for ℓ_{tar}) are not neighbors.

Observation 3: If ℓ_i is a non-corrupted machine, then it does not output msg.

Sketch of the proof: Before outputting msg, ℓ_i verifies that it is on the list $(\ell_1, \ldots, \ell_p)$ and that ℓ_{i+1} is its neighbor. Since the latter does not hold, ℓ_i drops msg. ■

Observation 4: If ℓ_{i+1} is a non-corrupted machine, then it does not output msg.

Sketch of the proof: The proof of this is similar to that of Observation 3. ■

In summary, we know that ℓ_{tar} has output msg, where msg carries the list of machines $(\ell_1, \ldots, \ell_p)$. We also know that there must be an $1 \leq i \leq p$ such that ℓ_i and ℓ_{i+1} are not neighbors. In addition, if ℓ_1, ..., ℓ_p, and ℓ_{tar} are all non-corrupted machines, then neither ℓ_i nor ℓ_{i+1} has output msg. The question is then how msg could reach ℓ_{ini} from ℓ_{tar}? The key observation is that A must have output msg. Can A bridge the gap between ℓ_i and ℓ_{i+1}? This is possible if A overhears the transmission of msg by a machine ℓ_x for some $x > i+1$ and can transmit msg to another machine ℓ_y for some $y < i$.

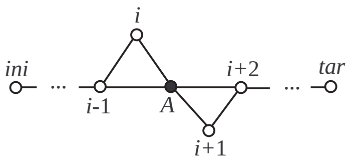

Figure 5: Part of a configuration where an attack against SRP is possible

Attack. Let us consider Figure 5, which illustrates part of a configuration where an attack against SRP based on the above observations is possible. The attack scenario is the following: ℓ_{ini} sends a route request towards ℓ_{tar}. The request reaches ℓ_{i-1} that re-broadcasts it. Thus, A receives the following route request message:

$$msg_1 = (\mathsf{rreq},\ \ell_{ini},\ \ell_{tar},\ id,\ sn,\ mac_{\ell_{ini}},\ (\ell_1, \ldots, \ell_{i-1}))$$

A then broadcasts the following message in the name of ℓ_{i+1}:

$$msg_2 = (\mathsf{rreq},\ \ell_{ini},\ \ell_{tar},\ id,\ sn,\ mac_{\ell_{ini}},\ (\ell_1, \ldots, \ell_{i-1}, \ell_i, \lambda, \ell_{i+1}))$$

where λ is an arbitrary sequence of identifiers. Since ℓ_{i+2} is a neighbor of A, it will hear the transmission. In addition, since the list of machines in the message ends with ℓ_{i+1}, which is also a neighbor of ℓ_{i+2}, it will process the request and re-broadcast it. Later, ℓ_{tar} sends the following route reply back to ℓ_{ini}:

$$msg_3 = (\mathsf{rrep},\ \ell_{ini},\ \ell_{tar},\ id,\ sn,\ (\ldots, \ell_{i-1}, \ell_i, \lambda, \ell_{i+1}, \ell_{i+2}, \ldots),\ mac_{\ell_{tar}})$$

When ℓ_{i+2} sends this message to ℓ_{i+1}, A overhears the transmission, and forwards the message to ℓ_{i-1} in the name of ℓ_i. ℓ_{i-1} will accept the message and passes it on towards ℓ_{ini}. Finally, ℓ_{ini} will output the route $(\ell_{ini}, \ldots, \ell_i, \lambda, \ell_{i+1}, \ldots, \ell_{tar})$, which is clearly a non-existent route.

Note that when A generates m_2, it cannot be sure that ℓ_{i-1} and ℓ_i are neighbors. Similarly, it does not know if ℓ_{i+1} and ℓ_{i+2} are neighbors. Hence the attack may fail. However, the success probability of the attack is non-negligible, given that ℓ_{i-1}, ℓ_i, ℓ_{i+1}, and ℓ_{i+2} are all neighbors of A, and it is known that in this case, the probability that ℓ_{i-1} and ℓ_i, as well as ℓ_{i+1} and ℓ_{i+2} are also neighbors is significantly higher than if we just put these nodes on the plane randomly.

4.2 Analysis of Ariadne with signatures

Operation of Ariadne with signatures. Ariadne has been proposed in [13] 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 [22], on digital signatures, or on MACs. We discuss Ariadne with digital signatures.

S	:	$h_{\mathsf{S}} = MAC_{\mathsf{SD}}(\mathsf{rreq}, \mathsf{S}, \mathsf{D}, id)$
$\mathsf{S} \rightarrow *$	:	$(\mathsf{rreq}, \mathsf{S}, \mathsf{D}, id, h_{\mathsf{S}}, (), ())$
B	:	$h_{\mathsf{B}} = H(\mathsf{B}, h_{\mathsf{S}})$
$\mathsf{B} \rightarrow *$	:	$(\mathsf{rreq}, \mathsf{S}, \mathsf{D}, id, h_{\mathsf{B}}, (\mathsf{B}), (sig_{\mathsf{B}}))$
C	:	$h_{\mathsf{C}} = H(\mathsf{C}, h_{\mathsf{B}})$
$\mathsf{C} \rightarrow *$	:	$(\mathsf{rreq}, \mathsf{S}, \mathsf{D}, id, h_{\mathsf{C}}, (\mathsf{B}, \mathsf{C}), (sig_{\mathsf{B}}, sig_{\mathsf{C}}))$
$\mathsf{D} \rightarrow \mathsf{C}$	:	$(\mathsf{rrep}, \mathsf{D}, \mathsf{S}, (\mathsf{B}, \mathsf{C}), (sig_{\mathsf{B}}, sig_{\mathsf{C}}), sig_{\mathsf{D}})$
$\mathsf{C} \rightarrow \mathsf{B}$	:	$(\mathsf{rrep}, \mathsf{D}, \mathsf{S}, (\mathsf{B}, \mathsf{C}), (sig_{\mathsf{B}}, sig_{\mathsf{C}}), sig_{\mathsf{D}})$
$\mathsf{B} \rightarrow \mathsf{S}$	:	$(\mathsf{rrep}, \mathsf{D}, \mathsf{S}, (\mathsf{B}, \mathsf{C}), (sig_{\mathsf{B}}, sig_{\mathsf{C}}), sig_{\mathsf{D}})$

Figure 6: Operation example of Ariadne with signatures and format of Ariadne messages. The initiator of the route discovery is S, the target is D, and the intermediate nodes are B and C. id is a randomly generated query identifier, H is a publicly known one-way hash function, and MAC_{SD} is a MAC function used with the key shared by S and D. sig_{B}, sig_{C}, and sig_{D} are digital signatures of B, C, and D, respectively. Each signature is computed over the message fields that precede the signature.

The operation of Ariadne with digital signatures is illustrated in Figure 6. There are two main differences between Ariadne and SRP. First, in Ariadne not only the initiator and the target authenticate the protocol messages, but intermediate nodes too insert their own digital signatures in route requests. Second, Ariadne uses per-hop hashing to prevent removal of identifiers from the accumulated route in the route request.

The initiator of the route discovery generates a route request message and broadcasts it to its neighbors. The route discovery message contains the identifiers of the initiator and the target, a randomly generated request identifier, and a MAC computed over these elements with a key shared by the initiator and the target. This MAC is hashed iteratively by each intermediate node together with its own identifier using a publicly known one-way hash function. The hash values computed in this way are called per-hop hash values. Each intermediate node that receives the request for the first time re-computes the per-hop hash value, appends its identifier to the list of identifiers accumulated in the request, and generates a digital signature on the updated request. Finally, the signature is appended to a signature list in the request, and the request is re-broadcasted.

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 all the digital signatures in the request. If all these verifications are successful, then the target generates a route reply and sends it back to the initiator via the reverse of the route obtained from the route request. The route reply contains the identifiers of the target and the initiator, the route and the list of digital signatures obtained from the request, and the digital signature of the target on all these elements. Each intermediate node passes the reply to the next node on the route (towards the initiator) without any modifications. When the initiator receives the reply, it verifies the digital signature of the target and the digital signatures of the intermediate nodes (for this it needs to reconstruct the requests that the intermediate nodes signed). If the verifications are successful, then it accepts the route returned in the reply.

We assume that every node performs the same verifications on the accumulated routes found in the routing messages as those described in the previous subsection in the context of SRP.

Analysis. Ariadne is not secure in our model. When attempting to derive a proof, one can make the following observations, which actually lead to the construction of an attack scenario described below.

Let us suppose that for some configuration $conf = (G, \tilde{v})$ and adversary A, the following message is received by a non-corrupted machine ℓ_{ini} in $sys^{\mathsf{ideal}}_{conf,A}$:

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

Let us further suppose that msg has been received with a corruption flag set to $\top$, and that msg passed all the verifications required by Ariadne at ℓ_{ini}. This means that all signatures in msg are correct, ℓ_1 is a neighbor of ℓ_{ini}, and $(\ell_{ini}, \ell_1, \ldots, \ell_p, \ell_{tar})$ is a non-existent route in G. We will denote the query identifier and the initiator's MAC that are used in the corresponding route request and needed for the verification of the signatures in msg by id and mac, respectively.

Observation 1: Given that the assumptions above hold, both adversary A and ℓ_{tar} must have output msg. This can be proven in the same way as in the case of SRP.

By assumption, ℓ_1 is a neighbor of ℓ_{ini}, and $(\ell_{ini}, \ell_1, \ldots, \ell_p, \ell_{tar})$ is a non-existent route. This means that there exists $1 \leq i \leq p$ such that ℓ_i and ℓ_{i+1} (where ℓ_{p+1} stands for ℓ_{tar}) are not neighbors.

Observation 2: $\ell_{i+1} = A$ with overwhelming probability.

Sketch of the proof: Assume that $\ell_{i+1} \neq A$. This means that ℓ_{i+1} is a non-corrupted machine. Since A can generate $sig_{\ell_{i+1}}$ only with negligible probability, ℓ_{i+1} must have generated and output the following request with overwhelming probability:

$$(\mathsf{rreq},\ \ell_{ini},\ \ell_{tar},\ id,\ h_{\ell_{i+1}},\ (\ell_1, \ldots, \ell_{i+1}),\ (sig_{\ell_1}, \ldots, sig_{\ell_{i+1}}))$$

where $h_{\ell_{i+1}} = H(\ell_{i+1}, H(\ell_i, H(\ldots H(\ell_1, mac))))$. However, ℓ_{i+1} cannot output this request, because it verifies the accumulated route in the request, and detects that the preceding machine ℓ_i is not its neighbor. ∎

Observation 3: A may be able to generate the following message without being the neighbor of ℓ_i:

$$(\mathsf{rreq},\ \ell_{ini},\ \ell_{tar},\ id,\ h_A,\ (\ell_1, \ldots, \ell_i, A),\ (sig_{\ell_1}, \ldots, sig_{\ell_i}, sig_A))$$

where $h_A = H(A, h_{\ell_i})$.

Sketch of the proof: A can easily generate this message if it receives $sig_{\ell_1}, \ldots, sig_{\ell_i}$, and any h_{ℓ_j} such that $j \leq i$ as parts of some other message. The key observation is that A does not necessarily need to receive these in a single message. Figure 7 illustrates a configuration, where A can receive all the necessary information to easily generate the message above. ∎

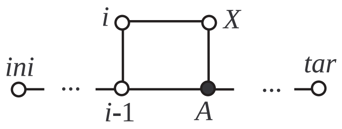

Figure 7: Part of a configuration where an attack against Ariadne is possible

Attack. The observations above lead to the following attack scenario (see Figure 7): ℓ_{ini} sends a route request towards ℓ_{tar}. The request reaches ℓ_{i-1} that re-broadcasts it. Thus, A receives the following route request message:

$$msg_1 = (\mathsf{rreq},\ \ell_{ini},\ \ell_{tar},\ id,\ h_{\ell_{i-1}},\ (\ell_1, \ldots, \ell_{i-1}), (sig_{\ell_1}, \ldots, sig_{\ell_{i-1}}))$$

A does *not* re-broadcast msg_1. Later, A receives another route request from X:

$$msg_2 = (\text{rreq},\ \ell_{ini},\ \ell_{tar},\ id,\ h_X,\ (\ell_1, \ldots, \ell_{i-1}, \ell_i, X), (sig_{\ell_1}, \ldots, sig_{\ell_{i-1}}, sig_{\ell_i}, sig_X))$$

From msg_2, A knows that ℓ_i is a neighbor of ℓ_{i-1}. A computes $h_A = H(A, H(\ell_i, h_{\ell_{i-1}}))$, where $h_{\ell_{i-1}}$ is obtained from msg_1. A obtains the signatures $sig_{\ell_1}, \ldots, sig_{\ell_i}$ from msg_2. Then, A generates and broadcasts the following request:

$$msg_3 = (\text{rreq},\ \ell_{ini},\ \ell_{tar},\ id,\ h_A,\ (\ell_1, \ldots, \ell_{i-1}, \ell_i, A), (sig_{\ell_1}, \ldots, sig_{\ell_{i-1}}, sig_{\ell_i}, sig_A))$$

Later, ℓ_{tar} generates the following route reply and sends it back towards ℓ_{ini}:

$$msg_4 = (\text{rrep},\ \ell_{tar},\ \ell_{ini},\ (\ell_1, \ldots, \ell_{i-1}, \ell_i, A, \ldots),\ (sig_{\ell 1}, \ldots, sig_{\ell i}, sig_A, \ldots),\ sig_{\ell_{tar}})$$

When A receives this route reply, it forwards it to ℓ_{i-1} in the name of ℓ_i. Finally, ℓ_{ini} will output the route $(\ell_{ini}, \ell_1, \ldots, \ell_{i-1}, \ell_i, A, \ldots, \ell_{tar})$, which is a non-existent route.

4.3 A provably secure routing protocol

Inspired by Ariadne, we present 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. The operation and the messages of endairA are illustrated in Figure 8.

$\mathsf{S} \rightarrow *$	:	$(\text{rreq},\ \mathsf{S},\ \mathsf{D},\ id,\ ())$
$\mathsf{B} \rightarrow *$	:	$(\text{rreq},\ \mathsf{S},\ \mathsf{D},\ id,\ (\mathsf{B}))$
$\mathsf{C} \rightarrow *$	:	$(\text{rreq},\ \mathsf{S},\ \mathsf{D},\ id,\ (\mathsf{B},\ \mathsf{C}))$
$\mathsf{D} \rightarrow \mathsf{C}$	:	$(\text{rrep},\ \mathsf{S},\ \mathsf{D},\ (\mathsf{B},\ \mathsf{C}),\ (sig_{\mathsf{D}}))$
$\mathsf{C} \rightarrow \mathsf{B}$	:	$(\text{rrep},\ \mathsf{S},\ \mathsf{D},\ (\mathsf{B},\ \mathsf{C}),\ (sig_{\mathsf{D}},\ sig_{\mathsf{C}}))$
$\mathsf{B} \rightarrow \mathsf{S}$	:	$(\text{rrep},\ \mathsf{S},\ \mathsf{D},\ (\mathsf{B},\ \mathsf{C}),\ (sig_{\mathsf{D}},\ sig_{\mathsf{C}},\ sig_{\mathsf{B}}))$

Figure 8: Operation example and messages of endairA. The initiator of the route discovery is S, the target is D, and the intermediate nodes are B and C. id is a randomly generated query identifier. sig_{B}, sig_{C}, and sig_{D} are digital signatures of B, C, and D, respectively. Each signature is computed over the message fields that precede the signature.

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 query identifier. Each intermediate node that receives the request for the first time appends its identifier to the route accumulated so far, 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 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 route carried by the reply, and that the preceding and following identifiers on the route belong to neighboring nodes. 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.

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

Sketch of the proof: In order to prove that endairA is statistically secure, it is enough to show that for any configuration $conf$ and any adversary A, a route reply message in $sys^{\mathsf{ideal}}_{conf,A}$ is dropped due to its corruption flag set to $\top$ with negligible probability.

Let us suppose that for some configuration $conf = (G, \tilde{v})$ and adversary A, the following message is received by a non-corrupted machine ℓ_{ini} in $sys^{\mathsf{ideal}}_{conf,A}$:

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

Let us further suppose that msg has been received with a corruption flag set to $\top$, and that msg passed all the verifications required by endairA at ℓ_{ini}. This means that all signatures in msg are correct, ℓ_1 is a neighbor of ℓ_{ini}, and $(\ell_{ini}, \ell_1, \ldots, \ell_p, \ell_{tar})$ is a non-existent route in G. It follows that there exists $1 \leq i \leq p$ such that ℓ_i and ℓ_{i+1} (where ℓ_{p+1} stands for ℓ_{tar}) are not neighbors.

We prove that the above is only possible if A forged the signature of ℓ_i or ℓ_{i+1}. (a) Let us assume that $\ell_i \neq A$. Then, ℓ_i is non-corrupted, and it verifies the route in the route reply before signing it. Consequently, it detects that ℓ_{i+1} is not its neighbor and it does not sign the route reply. As other non-corrupted machines do not generate signatures in the name of ℓ_i, ℓ_{ini} can receive msg only if A forged sig_{ℓ_i}. (b) Now let us assume that $\ell_i = A$. Then, ℓ_{i+1} is non-corrupted, and an argument similar to the one above leads to the conclusion that A must have forged $sig_{\ell_{i+1}}$.

It should be intuitively clear that if the signature scheme is secure, then A can forge a signature only with negligible probability, and thus, a route reply message in $sys^{\mathsf{ideal}}_{conf,A}$ is dropped due to its corruption flag set to $\top$ 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 A such that a route reply message in $sys^{\mathsf{ideal}}_{conf,A}$ is dropped due to its corruption flag set to $\top$ 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 on 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 $sys^{\mathsf{real}}_{conf,A}$ where all machines are initialized as described in the model, except that that the public key of a randomly selected non-corrupted machine 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 machines later, since the public verification key of ℓ_i is replaced with puk. By assumption, with probability ϵ, the simulation of $sys^{\mathsf{real}}_{conf,A}$ will result in a route reply message msg such that all signatures in msg are correct and msg contains a non-existent route. As we saw above, this means that there exists a non-corrupted machine 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. ■

Note 1: The proof uses only the fact that the adversary has only a single compromised key. In particular, the same proof would apply to an Active-1-x adversary, which has a single compromised key, but several devices in the network.

Note 2: While we designed endairA purely for demonstration purposes, it has some remarkable features. Besides being provably secure against an Active-1-1 adversary (and most probably against an Active-1-x adversary too), it is extremely simple and intuitive (e.g., it does not use per-hop hash values). In addition, it requires the nodes to sign only route reply messages, which means that the nodes need to produce orders of magnitude less signatures than in Ariadne, where the route request is signed by every node in the network due to the flooding of the request.

We must note, however, that endairA is not very resistant against DoS attacks. In particular, it allows the network to be flooded with fake route request messages. However, its resistance to such attacks can be increased by requiring the initiator to sign the request and the intermediate nodes to verify this signature.

5 Related work

There are several proposals for secure ad hoc routing protocols (see [15] for a recent overview). However, these proposals come with an informal security analysis with all the pitfalls of informal security arguments. Another set of papers deal with provable security for cryptographic algorithms and protocols (see Parts V and VI of [18] 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 [6], [26], and [23]. These papers apply the simulation paradigm for different security problems: [6] and [26] deal with key exchange protocols, and [23] 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 approach in the context of routing protocols for wireless ad hoc networks.

A different approach with similar goals to ours is presented in [28]. The authors of [28] 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 [10], 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. As we mentioned earlier, our definition corresponds to the informal definitions of security given in [21] and [13]. In addition, it seems to be impossible to guarantee that discovered routes do not contain corrupted nodes, since corrupted nodes can behave correctly and follow the routing protocol faithfully.

Another approach, presented in [19], 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 [19] is very much centered around the analysis of SRP, 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. The CPAL-ES analysis of SRP in [19] does not identify the attack presented in Section 4, which we have discovered with our approach. On the other hand, the advantage of the approaches of [19] and [28] is that they can be automated.

Finally, we must mention that SRP has been analyzed by its authors in [21] using BAN logic [7]. However, BAN logic has never been intended for the analysis of routing protocols. There are several problems with applying BAN logic in this context. First, BAN logic has no means to describe the goals of secure routing protocols, and to derive them from the protocol. It might be possible to extend BAN logic in such a way that those goals can be represented in it, but the authors of [21] have not done that. Second, a basic assumption of BAN logic is that the protocol participants are trustworthy and do not release secrets [8]. 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. It is dangerous to draw conclusions from a BAN analysis of the protocol when the basic assumptions of BAN logic are not satisfied. The fact that the BAN analysis of SRP in [21] was inappropriate is best illustrated by the attack presented in Section 4, which was completely overlooked by the authors of [21].

6 Conclusion and future work

In this paper, we made the first steps toward a formal model in which one can precisely define what secure routing means and prove (or fail to prove) that a given routing protocol indeed satisfies that definition (under some cryptographic assumptions). Our approach is based on the commonly known simulation paradigm for proving cryptographic protocols correct. The main contribution of the paper is the application of this approach to on-demand source routing protocols proposed for wireless ad hoc networks.

More specifically, we formally defined a real-world and an ideal-world model that capture the basic features of wireless ad hoc networking in general, and ad hoc routing protocols in particular. The real-world model describes the real operation of the routing protocol, while the ideal-world model formalizes the requirement that a secure routing protocol should not return non-existent routes to honest parties. Then, we gave a formal definition of routing security in terms of computational indistinguishability of the two models from the point of view of honest parties.

We demonstrated the usefulness of our approach by analyzing two “secure” ad hoc routing protocols, SRP and Ariadne. This analysis has led to the discovery of as yet unknown attacks against both protocols. Finally, we proposed a novel on-demand source routing protocol for wireless ad hoc networks, which can be proven to be secure in our model. This protocol served purely illustrative purposes in this paper, however, it has some remarkable features that make it worthy to consider by protocol designers when building their future protocols.

In terms of future work, we intend to extend our model to handle parallel protocol runs and Active-x-y adversaries (currently it handles only Active-1-1 adversaries). We also intend to adopt our model for routing protocols that use routing tables instead of source routes (e.g., SEAD [14] and ARAN [25]).

A particularly interesting direction for future work is trying to automate the analysis of ad hoc routing protocols. There are analysis approaches based on formal methods, such as state exploration tools and process calculi, that are easy to automate. However, they usually consider cryptographic primitives as unbreakable black boxes. This has the disadvantage of abstracting away details that may be important with respect to security. On the other hand, analysis approaches, such as the simulation approach, that do not abstract away cryptography provide rigorous results with respect to security, but they seem to be more difficult to automate. Recent work in both the formal method and the cryptography community has started to bridge this gap between the two approaches [17, 1, 11, 24]. Our intention is to extend these pioneering attempts to ad hoc routing protocols, and to define a model which is cryptographically sound but, at the same time, amenable to automation.

Acknowledgement

This work has partially been supported by the Hungarian Scientific Research Fund under project number T046664. The work of the first author have been further supported by the Hungarian Ministry of Education under contract number BÖ2003/70.

References

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proceedings of the IFIP Conference on Theoretical Computer Science*, Springer LNCS 1872, pages 3–22, 2000.

[2] M. Backes and B. Pfitzmann. A Cryptographically Sound Security Proof of the Needham-Schroeder-Lowe Public-Key Protocol. to appear in *IEEE Journal on Selected Areas in Communication*.

[3] D. Beaver. Foundations of secure interactive computing. In *Proceedings of Crypto'91*, 1991.

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proceedings of Crypto'93*, 1993.

[5] M. Bellare and P. Rogaway. Provably secure session key distribution – the three party case. In *Proceedings of the ACM Symposium on the Theory of Computing*, May 1995.

[6] 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.

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

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

[9] R. Canetti. Studies in Secure Multiparty Computation and Applications. PhD dissertation, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, June 1995.

[10] 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.

[11] J. Guttman, F. J. Thayer, and L. Zuck. The faithfulness of abstract protocol analysis: message authentication. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2001.

[12] Z. Haas, M. Perlman, and P. Samar. The Interzone Routing Protocol (IERP) for ad hoc networks. Internet Draft, IETF MANET Working Group, June 2001.

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

[14] Y.-C. Hu, D. Johnson, and A. Perrig. SEAD: Secure efficient distance-vector routing for mobile wireless ad hoc networks. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, 2002.

[15] 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.

[16] 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.

[17] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *Proceedings of Formal Methods'99*, Springer LNCS 1708, pages 776–793, 1999.

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

[19] 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.

[20] S. Micali and P. Rogaway. Secure computation. In *Proceedings of Crypto'91*, 1991.

[21] 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.

[22] 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.

[23] 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.

[24] B. Pfitzmann. Sound idealizations of cryptography for tool-supported proofs (position statement for panel discussion). In *Proceedings of the ACM Workshop on Formal Methods in Security Engineering (FMSE)*, Oct 2003.

[25] 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.

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

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

[28] 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.

A A note on the implicit deletion of routes

In principle, there are exactly two ways by which the effect of implicit deletion of routes (see the discussion on the ideal-world model in Subsection 2.1) can be prevented. One of them is when every node authenticates all incoming messages, and therefore dummy messages injected by the adversary are foiled immediately by one of the neighbors of the adversary. In this case the dummy messages do not change the state of any node in the network. The other way is when the intermediate nodes do not filter route request messages based on a request identifier, but they re-broadcast all requests received in a round and not heard previously. In this case, a dummy message injected by the adversary does not prevent the processing of other requests with the same request identifier as the dummy, and hence no route is deleted implicitly.

The first way is costly to implement, because it needs extensive cryptographic operations and corresponding security management. The second way is investigated in more details below.

The number of elementary operations required by a machine to perform its tasks in a round basically depends on the number of input messages to be processed and the number of output messages to be produced, where the amount of calculations per message is basically determined by the eventual cryptographic operations (e.g MAC verification, generation, etc.). The number of processed input messages depends on the communication graph (through the number of neighbors), while the number of output messages (per node per round) is specified by the protocol. If this latter number is limited by a constant t, then, obviously, the number of input messages will also be limited by tn, where n denotes the number of vertices of the graph. The other extreme is, when the number of output messages is not limited. In this case, nodes re-broadcast all received route requests after processing them according to the protocol.

Graph G can be represented by a binary string the length of which is a polynomial in the number of nodes (e.g., the upper half of the adjacency matrix). The security parameter k, typically, gives the size of some cryptographic material (e.g., the binary length of a secret MAC key). Let $T(G,k)$ denote the amount of time required to run the protocol summed over nodes and rounds. The processing time of a message per node per round is bounded by a polynomial $proc(n,k)$ (typically $b \cdot n \cdot k^{c_1}$ for some constants $b, c_1 > 0$). The amount of communication $com(G)$ is defined as the total number of input/output messages summed over all nodes and rounds and averaged over randomly selected source-target pairs. If the number of output messages per node per round is limited by constant t, then $com(G) = \mathcal{O}(n^{c_2})$ for some constant $c_2 > 0$. However, for an all-forwarding protocols, $com(G) = \mathcal{O}(e^{c_3 \cdot n})$ for some constant $c_3 > 0$ for the majority of randomly selected graphs (i.e., the total number of forwarded messages exponentially increases in the number nodes).

$T(G,k)/n = com(G) \cdot proc(n,k)/n$ gives the amount of time per node per run (averaged over source-target pairs). All participants use polynomial time machines, therefore such machines have not enough resources to run an all-forwarding protocol in a typical communication graph. Consequently we have to require a finite limit on the number of output messages. Note that an exponential-time resource could perform a successful brute force attack against cryptographic primitives.

Corollary: In case of polynomial time machines, implicit deletion of routes can only be prevented by requiring every node to authenticate all incoming messages.

Note: It is a plausible expectation that setting parameter t to higher fixed value would decrease the effect of implicit deletion: the probability that the number of messages arriving to a node remains below t increases. It might happen that keeping t as high as the implementation cost can afford could decrease the effect of implicit deletion to a practically acceptable level, however, we cannot expect asymptotical negligibility.

B An Active-1-2 attack against Ariadne

In this section, we present another attack against Ariadne that can be mounted by an Active-1-2 adversary. Let us consider the configuration illustrated in Figure 9. The adversary has two devices but only a single corrupted key (identity A). We explain the attack when Ariadne is used with standard MACs, but it also works if TESLA is used, or when signatures are used and intermediate nodes do not verify the signature list in the route request.

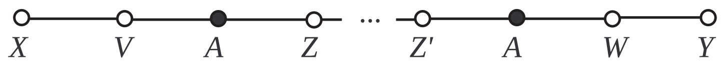

Figure 9: Part of a configuration where an Active-1-2 attack against Ariadne is possible

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

$$msg_1 = (\mathsf{rreq},\ X,\ Y,\ id,\ h_V,\ (V), (mac_{VY}))$$

A does not append his MAC to the request, instead, it puts h_V on the MAC list, and re-broadcasts the following request:

$$msg_2 = (\mathsf{rreq},\ X,\ Y,\ id,\ h_V,\ (V, A), (mac_{VY}, h_V))$$

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

$$msg_3 = (\mathsf{rreq},\ X,\ Y,\ id,\ H(Z', H(Z, h_V)),\ (V, A, Z, Z'), (mac_{VY}, h_V, mac_{ZY}, mac_{Z'Y}))$$

A removes Z and Z' from the node list and their MACs from the MAC list[6]. A can do this in the following way: By noticing identifier A in the accumulated route, A knows that the request passed through the first adversarial node. By looking at the position of identifier A in the node list, A will know where h_V is on the MAC list. From h_V, A computes $h_A = H(A, h_V)$ and a MAC on $(\mathsf{rreq}, X, Y, id, h_A, (V, A), mac_{VY})$, and re-broadcasts the following request:

$$msg_4 = (\mathsf{rreq},\ X,\ Y,\ id,\ h_A,\ (V, A), (mac_{VY}, mac_{AY}))$$

As the per-hop hash value and all the MACs are correct in msg_4, Y will receive a correct request, and returns the following reply:

$$msg_5 = (\mathsf{rrep},\ Y,\ X,\ (V, A, W),\ mac_{YX})$$

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

$$msg_6 = (\mathsf{rrep},\ Y,\ X,\ (V, A, Z, Z', A, W),\ mac_{YX})$$

Note that neither Z' nor Z can 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,

[6]If there are more nodes on the route between the two adversarial nodes, then all of them can be removed.

both of them forwards the reply. Finally, when the first adversarial node receives the reply, it removes Z, Z', and A from the node list:

$$msg_7 = (\mathsf{rrep},\ Y,\ X,\ (V, A, W),\ mac_{YX})$$

In this way, X receives the route reply that Y sent. This means that X accepts the route (X, V, A, W, Y), which is non-existent.