Analysing routing protocols: four nodes topologies are sufficient*

Véronique Cortier[1], Jan Degrieck[1,2], and Stéphanie Delaune[2]

[1] LORIA, CNRS, France
[2] LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France, France

Abstract. Routing protocols aim at establishing a route between nodes on a network. Secured versions of routing protocols have been proposed in order to provide more guarantees on the resulting routes. Formal methods have proved their usefulness when analysing standard security protocols such as confidentiality or authentication protocols. However, existing results and tools do not apply to routing protocols. This is due in particular to the fact that all possible topologies (infinitely many) have to be considered.

In this paper, we propose a simple reduction result: when looking for attacks on properties such as the validity of the route, it is sufficient to consider topologies with only four nodes, resulting in a number of just five distinct topologies to consider. As an application, we analyse the SRP applied to DSR and the SDMSR protocols using the ProVerif tool.

1 Introduction

Routing protocols aim at establishing a route between distant nodes on a network. Attacking routing protocols is a first step towards mounting more sophisticated attacks. For example, forcing a route to visit a malicious node allows an attacker to monitor and listen to the traffic, possibly blocking some messages. Therefore, secured versions of routing protocols have been proposed to provide more guarantees on the resulting routes, but they are often still subject to attacks. For example, the SRP protocol [26] is a generic construction for securing protocols. However, applied to DSR [23], a standard routing protocol, it has been shown to be flawed, allowing an attacker to modify the route, making the source node to accept an invalid route [15]. This shows that the design of secure routing protocols is difficult and error-prone.

In the context of standard security protocols such as confidentiality or authentication protocols, formal methods have proved their usefulness for providing security guarantees or detecting attacks. For example, a flaw has been discovered (see [5]) in the Single-Sign-On protocol used *e.g.* by Google Apps. It has

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 258865, project ProSecure, and the ANR project JCJC VIP n^o 11 JS02 006 01.

been shown that a malicious application could very easily access to any other application (*e.g.* Gmail or Google Calendar) of their users. This flaw has been found when analyzing the protocol using formal methods, abstracting messages by a term algebra and using the AVISPA platform [6]. More generally, many decision procedures (*e.g.* [27, 12]) have been proposed for automatically analyzing the security of protocols. Tools like AVISPA, ProVerif [13], or Scyther [18] have been developed and successfully applied to protocols, yielding discoveries of attacks or security proofs.

However, these results and tools do not apply to routing protocols. One of the main reasons is the fact that analysing routing protocols requires one to consider a different attacker model. Indeed, in contrast to standard security protocols where the attacker is assumed to control all the communications, an attacker for routing protocols is *localized*, *i.e.* it can control only a finite number of nodes (typically one or two). Since a node broadcasts its messages to all its neighbours, it is very easy for a malicious node to listen to the communication of its neighbours but it is not possible to listen beyond the neighbouring nodes. Therefore, the existence of an attack strongly depends on the network topology, that is, how nodes are connected and where malicious nodes are located.

Some dedicated techniques have been developed for formally analyzing routing protocols. For example, S. Nanz and C. Hankin [25] have proposed one of the first formal models for routing protocols and have shown how to automatically analyze a finite number of attack scenarios. For a general attacker, M. Arnaud *et al.* [7] have proposed an NP decision procedure for a finite number of sessions. Several case studies have also been conducted. For example, D. Benetti *et al.* [10] have analyzed the ARAN and endairA protocols with the AVISPA tool, considering a finite number of scenarios. G. Ács *et al.* [3] have developed a framework for analyzing the distance vector routing protocols SAODV and ARAN. However, these results are rather ad-hoc and no decision procedure has been implemented.

Our contribution. Instead of proposing a new decision procedure, we propose in this paper a simple reduction result: if there is an attack, then there is an attack on a small network topology with only four nodes. More precisely, we show that at most five distinct topologies (each with four nodes) need to be considered when looking for an attack. We therefore reduce the number of topologies to be considered from infinitely many to only five. Our reduction result holds for properties such as route validity and for a very general class of routing protocols. Indeed, we consider arbitrary cryptographic primitives (provided they can be expressed as terms) and arbitrary protocol transitions. For example, our model allows neighbourhood tests, recursive operations, and of course standard pattern-matching, encompassing the models proposed in [25, 7, 8].

The proof of our reduction result consists in two main steps. First, we show that if there is an attack, then the attack is preserved when adding all but one edge to the network topology, yielding a *quasi-complete graph.* Second, we show how to merge all the nodes having the same neighbourhood and honesty/dishonesty status. It is then sufficient to observe that merging quasi-complete graphs results into only five distinct topologies, each of them containing four nodes.

An interesting consequence of our reduction result is that it allows one to reuse techniques and tools developed for standard security protocols. Indeed, it is now possible to consider the five fixed topologies one by one and to analyse the protocol in each of the five cases using existing tools, provided of course that the protocol's primitives are supported by the tool. As an application, we analyse the SRP applied to DSR [23, 26] and the SDMSR [20] protocols using the ProVerif tool, retrieving the existing attacks. Detailed proofs of our results can be found in [17].

Related work. Our result follows the spirit of [16] where it is shown that only two distinct identities need to be considered when studying confidentiality or authentication properties. To our knowledge, the only approach proposing a reduction result in the context of routing protocols is [4]. In this paper, the authors show how to reduce the number of network topologies that need to be considered, taking advantage of the symmetries. However, the total number of configurations is still infinite in the general case or really large even when considering a bounded number of nodes. For example, more than 30000 topologies need to be considered when the number of nodes is bounded by six. In contrast, our result reduces to only five topologies, even when considering attacks with arbitrarily many nodes.

2 Messages and attacker capabilities

For modeling messages, we consider an arbitrary term algebra and deduction system, which provides a lot of flexibility in terms on which cryptographic primitives can be modeled.

2.1 Messages

Messages are represented by terms where cryptographic primitives such as encryption, signature, and hash function, are represented by *function symbols*. More precisely, we consider a *signature* $(\mathcal{S}, \Sigma)$ made of a set of sorts $\mathcal{S}$ and a set of function symbols Σ together with arities of the form $ar(f) = s_1 \times \ldots \times s_k \to s$ where $f \in \Sigma$, and $s, s_1, \ldots, s_k \in \mathcal{S}$. We consider an infinite set of *variables* $\mathcal{X}$ and an infinite set of *names* $\mathcal{N}$ that typically represent nonces, session keys, or agent names. We assume that names and variables are given with sorts.

Regarding the sort system, we consider a special sort agent that only contains names and variables. These names represent the names of the agents, also called the nodes of the network. We assume a special sort term that subsumes all the other sorts and such that any term is of sort term. Terms are defined as names, variables, and function symbols applied to other terms. Of course function symbol application must respect sorts and arities. For $\mathcal{A} \subseteq \mathcal{X} \cup \mathcal{N}$, the set of terms built from $\mathcal{A}$ by applying function symbols in Σ is denoted by $\mathcal{T}(\Sigma, \mathcal{A})$. A term t is said to be a *ground* term if it does not contain any variable.

Example 1. A typical signature for representing the primitives used in routing protocols such as the SRP [26] protocol is the signature $(\mathcal{S}_{\mathsf{SRP}}, \Sigma_{\mathsf{SRP}})$ defined by $\mathcal{S}_{\mathsf{SRP}} = \{\mathsf{agent}, \mathsf{list}, \mathsf{term}\}$ and $\Sigma_{\mathsf{SRP}} = \{hmac, \langle\rangle, ::, \bot, shk, req, rep\}$, with the following arities:

- $hmac : \mathsf{term} \times \mathsf{term} \to \mathsf{term}$,
- $\langle\rangle : \mathsf{term} \times \mathsf{term} \to \mathsf{term}$,
- $shk : \mathsf{agent} \times \mathsf{agent} \to \mathsf{term}$,
- $:: : \mathsf{agent} \times \mathsf{list} \to \mathsf{list}$,
- $\bot : \ \to \mathsf{list}$,
- $req, rep : \ \to \mathsf{term}$.

The symbol :: is the list constructor whereas $\bot$ is a constant representing an empty list. The constants *req* and *rep* are used to identify the request phase and the reply phase. The term $shk(A, B)$ $(= shk(B, A))$ represents a shared key between A and B. The term $hmac(m, k)$ represents the keyed hash message authentication code computed over message m with key k while $\langle\rangle$ is a pairing operator. We write $\langle t_1, t_2, t_3\rangle$ for the term $\langle t_1, \langle t_2, t_3\rangle\rangle$, and $[t_1; t_2; t_3]$ for $t_1 :: (t_2 :: (t_3 :: \bot))$.

Substitutions are written $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ with $dom(\sigma) = \{x_1, \ldots, x_n\}$, and $img(\sigma) = \{t_1, \ldots, t_n\}$. We only consider *well-sorted* substitutions, that is substitutions for which x_i and t_i have the same sort. The substitution σ is *ground* if the t_i are ground. The application of a substitution σ to a term t is written $\sigma(t)$ or $t\sigma$. A *most general unifier* of two terms t and u is a substitution denoted by $mgu(t, u)$. We write $mgu(t, u) = \sharp$ when t and u are not unifiable.

2.2 Attacker capabilities

The ability of the attacker is modeled by a deduction relation $\vdash \subseteq 2^{\mathsf{term}} \times \mathsf{term}$. The relation $\mathcal{I} \vdash v$ represents the fact that the term v is computable from the set of terms $\mathcal{I}$. Such a relation is defined through an inference system, *i.e.* a finite set of rules of the form $\dfrac{u_1 \ldots u_n}{u}$ where $u, u_1, \ldots, u_n \in \mathcal{T}(\Sigma, \mathcal{X})$. The deduction relation can be arbitrary in our model as long as the terms $u, u_1, \ldots u_n$ do not contain any names. An example of such a relation is provided below.

A term u is *deducible* from a set of terms $\mathcal{I}$, denoted by $\mathcal{I} \vdash u$, if there exists a proof, *i.e.* a tree such that the root is labelled with u and the leaves are labelled with $v \in \mathcal{I}$ and every intermediate node is an instance of one of the rules of the inference system.

Example 2. We can associate to the term algebra $(\mathcal{S}_{\mathsf{SRP}}, \Sigma_{\mathsf{SRP}})$ defined in Example 1, the following inference system.

$$\frac{y_1 \quad y_2}{\langle y_1, y_2\rangle} \qquad \frac{\langle y_1, y_2\rangle}{y_1} \qquad \frac{\langle y_1, y_2\rangle}{y_2} \qquad \frac{x \quad z}{x :: z} \qquad \frac{x :: z}{x} \qquad \frac{x :: z}{z} \qquad \frac{y_1 \quad y_2}{hmac(y_1, y_2)}$$

The terms y_1, y_2 are variables of sort term, x is a variable of sort agent, whereas z is a variable of sort list. This inference system reflects the ability for the attacker to concatenate terms, to build lists, and to retrieve components of lists and pairs.

The last inference rule models the fact that an attacker can also compute a MAC when he knows the key (and the message to be MACed).

Let $\mathcal{I} = \{S, D, A_1, A_2, \perp, \langle req, S, D, id, [S], hmac(\langle req, S, D, id\rangle, shk(S, D))\rangle\}$ and $m = \langle req, S, D, id, [A_1; A_2; S], hmac(\langle req, S, D, id\rangle, shk(S, D))\rangle$ where S, D, A_1, and A_2 are names of sort agent. The term m typically represents a message that the attacker would like to send over the network while $\mathcal{I}$ represents its knowledge so far, typically having listened to the first step of the SRP protocol. Considering the inference system described above, we have that $\mathcal{I} \vdash m$.

2.3 Functions over terms

In order to be as general as possible, we consider protocols that perform any operation on the terms they receive. We therefore consider *functions over terms*, that is, functions of the form $\mathsf{f} : \mathcal{T}(\Sigma, \mathcal{N}) \times \ldots \times \mathcal{T}(\Sigma, \mathcal{N}) \to \mathcal{T}(\Sigma, \mathcal{N})$.

These functions can of course model standard applications of cryptographic operations. For example, the function $(x, y, z) \mapsto \{\langle x, y\rangle\}_z$ represent the function that concatenates the terms x and y and then encrypts it with z. They can also be used to model various operations on lists. For instance, we can define the reverse function that takes as input a list $[A_1, \ldots, A_n]$ and outputs the reversed list $[A_n, \ldots, A_1]$.

More interestingly, such functions encompass recursive operations and recursive tests. Typical examples can be found in the context of routing protocols, when nodes check for the validity of the route. For example, in the SMNDP protocol [20], a route from the source A_0 to the destination A_n is represented by a list $l_{route} = [A_1; \ldots; A_n]$. This list is accepted by the source node A_0 only if the received message is of the form:

$$[[\![\langle A_n, A_0, l_{route}\rangle]\!]_{sk(A_1)}; [\![\langle A_n, A_0, l_{route}\rangle]\!]_{sk(A_2)}; \ldots; [\![\langle A_n, A_0, l_{route}\rangle]\!]_{sk(A_n)}]$$

where $[\![\langle A_n, A_0, l_{route}]\!]_{sk(A_i)}$ is a signature performed by A_i. This test and many others (*e.g.* [21, 15]) can be easily modelled using functions over terms. Clearly, not all functions over terms are meaningful to model protocols. In particular, some of them might not be executable. Actually, a precise definition of executability is not relevant for our result: our result holds for non executable functions as soon as they satisfy the properties stated in our Theorem 1.

3 Models for protocols

Several calculi have been proposed to model security protocols (*e.g.* [2, 1]). However, they are not well-adapted for routing protocols. For instance, in contrast to standard security protocols, the attacker is localized to some nodes and cannot control all the communications. The nodes, *i.e.* the processes, have to perform some specific actions that can not be easily modeled in such calculi, like recursive checks (checking a chain of signatures) or some sanity checks on the routes they receive, such as neighbourhood properties.

Actually, our calculus is inspired from CBS# [25] and generalize the calculus given in [7] by allowing processes to perform any operation on the terms they receive and considering an arbitrary signature for terms.

3.1 Syntax

The intended behavior of each node of the network can be modeled by a *process* defined by the grammar given below. Our calculus is parametrized by a set $\mathcal{P}$ of predicates and a set $\mathcal{F}$ of functions over terms, whose purpose is to represent the computations performed by the agents. We assume that these functions are total and deterministic. This means that a partial function will be modeled by returning a special constant *fail* when it is needed.

$\Phi, \Phi_1, \Phi_2 :=$	Formula	
$p(u_1, \dots, u_n)$	literal	with $p \in \mathcal{P}$
$\Phi_1 \wedge \Phi_2$	conjunction	
$P, Q, R :=$ Processes		
0	null process	
$\mathsf{out}(\mathsf{f}(u_1, \dots, u_n)).P$	emission	
$\mathsf{in}(u).P$	reception	
$\mathsf{if}\ \Phi\ \mathsf{then}\ P$	conditional	
$P \mid Q$	parallel composition	
$!P$	replication	
$\mathsf{new}\ n.P$	fresh name generation	

where $u, u_1, \dots, u_n$ are terms that may contain variables, n is a name, $\mathsf{f} \in \mathcal{F}$, and Φ is a formula.

Fig. 1. Syntax of processes

The process "$\mathsf{in}(u).P$" expects a message m of the form u and then behaves like $P\sigma$ where $\sigma = mgu(m, u)$. The process "$\mathsf{out}(\mathsf{f}(u_1, \dots, u_n)).P$" computes the term $u = \mathsf{f}(u_1, \dots, u_n)$, emits u, and then behaves like P. The purpose of $\mathsf{f}(u_1, \dots, u_n)$ is to model any operation f on the terms $u_1, \dots, u_n$ (the variables in $u_1, \dots, u_n$ will be instantiated when the evaluation will take place). For instance, such a function f can be used to reverse a list, or to apply some cryptographic primitives on top of $u_1, \dots, u_n$, or any combination of these operations. The process "$\mathsf{if}\ \Phi\ \mathsf{then}\ P$" behaves like P when Φ is true and stops otherwise.

We assume that the predicates $p \in \mathcal{P}$ are given together with their semantics that may depend on the underlying graph G that models the topology of the network. Such a graph $G = (V, E)$ is given by a set of vertices $V \subseteq \{A \mid A \in \mathcal{N} \text{ of sort } \mathsf{agent}\}$ and a set of edges $E \subseteq V \times V$. Since the purpose of this graph is to model the communication network, we consider topologies where E is a reflexive and symmetric relation. We consider two kinds of predicates: a set $\mathcal{P}_I$ of predicates whose semantics is independent of the graph, *i.e.* $[\![p(u_1, \dots, u_k)]\!]_G =$

$[\![p(u_1, \ldots, u_k)]\!]_{G'}$ for any graphs G and G' and any ground terms $u_1, \ldots, u_k$; and a set $\mathcal{P}_D$ of predicates whose semantics is dependent on the graph. The semantics of a formula is then defined as expected. The purpose of $\mathcal{P}_D$ is to model neighbourhood checks that are typically performed in routing protocols.

Example 3. As an illustrative purpose, we consider the set $\mathcal{P}_{\mathsf{SRP}} = \mathcal{P}_I \cup \mathcal{P}_D$ where $\mathcal{P}_I = \{\mathsf{first}, \mathsf{last}\}$ and $\mathcal{P}_D = \{\mathsf{check}, \mathsf{checkl}\}$. The purpose of the predicates in $\mathcal{P}_I$ is to model some sanity checks that are performed by the source when it receives the path. The semantics of these predicates is independent of the graph and is defined as follows:

- $\mathsf{first}(A, l) = \mathsf{true}$ if and only if l is of sort list and its first element is A;
- $\mathsf{last}(A, l) = \mathsf{true}$ if and only if l is of sort list and its last element is A.

The purpose of the predicates in $\mathcal{P}_D$ is to model neighbourhood checks. Given a graph $G = (V, E)$, their semantics is defined as follows:

- $\mathsf{check}(A, B)$ checks for neighbourhood of two nodes, $[\![\mathsf{check}(A, B)]\!]_G = \mathsf{true}$ if and only if $(A, B) \in E$, with A, B of sort agent;
- $\mathsf{checkl}(C, l)$ checks for local neighbourhood of a node in a list, $[\![\mathsf{checkl}(C, l)]\!]_G = \mathsf{true}$ if and only if C is of sort agent, l is of sort list, and for any l' subterm of l, if $l' = A :: C :: l_1$, then $(A, C) \in E$; whereas if $l' = C :: B :: l_1$, then $(C, B) \in E$.

We write $\mathit{fv}(P)$ for the set of *free variables* that occur in P, *i.e.* the set of variables that are not in the scope of an input. We consider *ground processes*, *i.e.* processes P such that $\mathit{fv}(P) = \emptyset$, and *parametrized* processes, denoted $P(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are variables of sort agent, and such that $\mathit{fv}(P) \subseteq \{x_1, \ldots, x_n\}$. A *routing role* is a parametrized process that do not contain any name of sort agent. A *routing protocol* is then simply a set of routing roles.

3.2 Example: modeling the SRP protocol

We consider the secure routing protocol SRP applied on DSR introduced in [26], assuming that each node already knows his neighbours (running *e.g.* some neighbour discovery protocol). SRP is not a routing protocol by itself, it describes a generic way for securing source-routing protocols. We model here its application to the DSR protocol [23]. DSR is a protocol which is used when an agent S (the source) wants to communicate with another agent D (the destination), which is not his immediate neighbour. In an ad hoc network, messages can not be sent directly to the destination, but have to travel along a path of nodes.

To discover a route to the destination, the source constructs a request packet and broadcasts it to its neighbours. The request packet contains its name S, the name of the destination D, an identifier of the request id, a list containing the beginning of a route to D, and a hmac computed over the content of the request with a key $\mathit{shk}(S, D)$ shared by S and D. It then waits for an answer containing a route to D with a hmac matching this route, and checks that it is a plausible route by checking for instance that his neighbour in the route is indeed a neighbour of S in the network.

Consider the signature given in Example 1, the predicates $\mathcal{P}_{\mathsf{SRP}}$ introduced in Example 3, and the set $\mathcal{F}_{\mathsf{SRP}}$ of functions over terms that only contains the identity function (for sake of clarity, we omit it). Let id be a name, x_S, x_D be variables of sort agent, and x_L be a variable of sort list. The process executed by the agent x_S initiating the search of a route towards a node x_D is:

$$P_{\mathsf{src}}(x_S, x_D) = \mathsf{new}\ id.\mathsf{out}(u_1).\mathsf{in}(u_2).\mathsf{if}\ \Phi_S\ \mathsf{then}\ 0$$

$$\text{where } \begin{cases} u_1 = \langle req, x_S, x_D, id, x_S :: \bot, hmac(\langle req, x_S, x_D, id\rangle, shk(x_S, x_D))\rangle \\ u_2 = \langle rep, x_D, x_S, id, x_L, hmac(\langle rep, x_D, x_S, id, x_L\rangle, shk(x_S, x_D))\rangle \\ \Phi_S = \mathsf{checkl}(x_S, x_L) \wedge \mathsf{first}(x_D, x_L) \wedge \mathsf{last}(x_S, x_L) \end{cases}$$

The names of the intermediate nodes are accumulated in the route request packet. Intermediate nodes relay the request over the network, except if they have already seen it. An intermediate node also checks that the received request is locally correct by verifying whether the head of the list in the request is one of its neighbours. Below, x_V, x_S, x_D and x_A are variables of sort agent whereas x_r is a variable of sort list and x_{id}, x_m are variables of sort term. The process executed by an intermediary node x_V when forwarding a request is as follows:

$$P_{\mathsf{request}}(x_V) = \mathsf{in}(w_1).\mathsf{if}\ \Phi_V\ \mathsf{then}\ \mathsf{out}(w_2).0$$

$$\text{where } \begin{cases} w_1 = \langle req, x_S, x_D, x_{id}, x_A :: x_r, x_m\rangle \\ \Phi_V = \mathsf{check}(x_V, x_A) \\ w_2 = \langle req, x_S, x_D, x_{id}, x_V :: (x_A :: x_r), x_m\rangle \end{cases}$$

When the request reaches the destination D, it checks that the request has a correct hmac and that the first node in the route is one of his neighbours. Then, the destination D constructs a route reply, in particular it computes a new hmac over the route accumulated in the request packet with $shk(x_S, D)$, and sends the answer back over the network.The process executed by the destination node x_D is the following:

$$P_{\mathsf{dest}}(x_D) = \mathsf{in}(v_1).\mathsf{if}\ \Phi_D\ \mathsf{then}\ \mathsf{out}(v_2).0$$

where:

$$\begin{cases} v_1 = \langle req, x_S, x_D, x_{id}, x_A :: x_l, hmac(\langle req, x_S, x_D, x_{id}\rangle, shk(x_S, x_D))\rangle \\ \Phi_D = \mathsf{check}(x_D, x_A) \\ v_2 = \langle rep, x_D, x_S, x_{id}, l_{route}, hmac(\langle rep, x_D, x_S, x_{id}, l_{route}\rangle, shk(x_S, x_D))\rangle \\ l_{route} = x_D :: x_A :: x_l \end{cases}$$

Then, the reply travels along the route back to x_S. The intermediary nodes check that the route in the reply packet is locally correct (that is that the nodes before and after them in the list are their neighbours) before forwarding it. The process executed by an intermediary node x_V when forwarding a reply is the following:

$$P_{\mathsf{reply}}(x_V) = \mathsf{in}(w').\mathsf{if}\ \Phi'_V\ \mathsf{then}\ \mathsf{out}(w')$$

where $w' = \langle rep, x_D, x_S, x_{id}, x_r, x_m\rangle$, and $\Phi'_V = \mathsf{checkl}(x_V, x_r)$.

Example 4. In our model, the routing protocol SRP is defined by the following set of parametrized processes:

$$\{P_{\mathsf{src}}(x_S, x_D);\ P_{\mathsf{request}}(x_V);\ P_{\mathsf{reply}}(x_V);\ P_{\mathsf{dest}}(x_D)\}.$$

3.3 Configuration and topology

Each process is located at a specified node of the network. Unlike the classical Dolev-Yao model [19], the intruder does not control the entire network but can only interact with its neighbours. More specifically, we assume that the *topology* of the network is represented by a tuple $\mathcal{T} = (G, \mathcal{M}, S, D)$ where:

- $G = (V, E)$ is an undirected graph with $V \subseteq \{A \in \mathcal{N} \mid A \text{ of sort } \mathsf{agent}\}$, where an edge in the graph models the fact that two agents are neighbours. We only consider graphs such that $\{(A, A) \mid A \in V\} \subseteq E$ which means that an agent can receive a message sent by himself;
- $\mathcal{M}$ is a set of nodes that are controlled by the attacker. These nodes are called *malicious* whereas nodes *not in* $\mathcal{M}$ are called *honest*;
- S and D are two honest nodes that represent respectively the source and the destination for which we analyse the security of the routing protocol.

Note that our model is not restricted to a single malicious node. In particular, our results apply to the case of several compromised nodes that communicate (and therefore share their knowledge), using out-of-band resources or hidden channels (*e.g.* running other instances of the routing protocols).

A *configuration* of the network is a pair $(\mathcal{P}; \mathcal{I})$ where:

- $\mathcal{P}$ is a multiset of expressions of the form $\lfloor P \rfloor_A$ that represents the (ground) process P executed by the agent $A \in V$. We will write $\lfloor P \rfloor_A \cup \mathcal{P}$ instead of $\{\lfloor P \rfloor_A\} \cup \mathcal{P}$.
- $\mathcal{I}$ is a set of ground terms representing the messages seen by the malicious nodes as well as their initial knowledge.

Example 5. Continuing our modeling of SRP, a typical initial configuration for the SRP protocol is

$$K_0 = (\lfloor P_{\mathsf{src}}(S, D) \rfloor_S \mid \lfloor P_{\mathsf{dest}}(D) \rfloor_D; \mathcal{I}_0)$$

where both the source node S and the destination node D wish to communicate. A more realistic configuration would include intermediary nodes but this initial configuration is already sufficient to present an attack. We assume that the initial knowledge of the intruder is given by a possibly infinite set of terms $\mathcal{I}_0$

$$\text{(Comm)}\quad \big(\{\lfloor \mathsf{in}(u_j).P_j\rfloor_{A_j} \mid (A,A_j)\in E \wedge\ mgu(t,u_j)\neq \sharp\}\cup \lfloor \mathsf{out}(\mathsf{f}(t_1,\ldots,t_n)).P\rfloor_A \cup \mathcal{P};\mathcal{I}\big) \rightarrow_{\mathcal{T}} \big(\{\lfloor P_j\sigma_j\rfloor_{A_j}\}\cup\lfloor P\rfloor_A\cup\mathcal{P};\mathcal{I}'\big)$$

$$\text{where } \begin{cases} \sigma_j = mgu(t,u_j) \text{ where } t \text{ is the result of applying } \mathsf{f} \text{ on } t_1,\ldots,t_n \\ \mathcal{I}' = \mathcal{I}\cup\{t\} \text{ if } (A,I)\in E \text{ for some } I\in\mathcal{M} \text{ and } \mathcal{I}'=\mathcal{I} \text{ otherwise.}\end{cases}$$

$$\text{(In)}\quad \big(\lfloor \mathsf{in}(u).P\rfloor_A\cup\mathcal{P};\mathcal{I}\big)\rightarrow_{\mathcal{T}}\big(\lfloor P\sigma\rfloor_A\cup\mathcal{P};\mathcal{I}\big)$$
$$\text{if } (A,I)\in E \text{ with } I\in\mathcal{M},\ \mathcal{I}\vdash t, \text{ and } \sigma = mgu(t,u)$$

$$\text{(Then)}\quad \big(\lfloor \mathsf{if}\ \Phi\ \mathsf{then}\ P\rfloor_A\cup\mathcal{P};\mathcal{I}\big)\rightarrow_{\mathcal{T}}\big(\lfloor P\rfloor_A\cup\mathcal{P};\mathcal{I}\big)\qquad \text{if } [\![\Phi]\!]_G = 1$$

$$\text{(Par)}\quad \big(\lfloor P_1\mid P_2\rfloor_A\cup\mathcal{P};\mathcal{I}\big)\rightarrow_{\mathcal{T}}\big(\lfloor P_1\rfloor_A\cup\lfloor P_2\rfloor_A\cup\mathcal{P};\mathcal{I}\big)$$

$$\text{(Repl)}\quad \big(\lfloor !P\rfloor_A\cup\mathcal{P};\mathcal{I}\big)\rightarrow_{\mathcal{T}}\big(\lfloor P\rfloor_A\cup\lfloor !P\rfloor_A\cup\mathcal{P};\mathcal{I}\big)$$

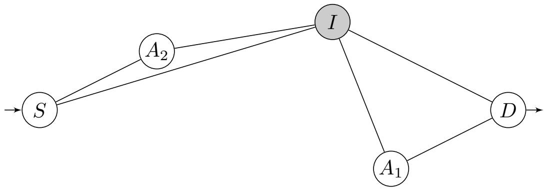
$$\text{(New)}\quad \big(\lfloor \mathsf{new}\ m.P\rfloor_A\cup\mathcal{P};\mathcal{I}\big)\rightarrow_{\mathcal{T}}\big(\lfloor P\{m\mapsto m'\}\rfloor_A\cup\mathcal{P};\mathcal{I}\big)$$
$$\text{where } m' \text{ is a fresh name}$$

where $\mathcal{T} = (G,\mathcal{M},S,D)$ and $G=(V,E)$.

Fig. 2. Transition system.

that typically contains the names of sort agent, the constants *req*, *rep*, and $\perp$, and the dishonest keys, *i.e.* those that belong to a malicious node.

A possible topology $\mathcal{T}_0 = (G_0,\mathcal{M}_0,S,D)$ is modeled by the graph G_0 below, where I is a malicious node, *i.e.* $\mathcal{M}_0 = \{I\}$ while A_1 and A_2 are two extra (honest) nodes.



3.4 Execution model

The communication system is formally defined by the rules of Figure 2. They are parametrized by the underlying topology $\mathcal{T}$.

The Comm rule allows nodes to communicate provided they are (directly) connected in the underlying graph. We do not assume that messages are necessarily delivered to the intended recipients. They may be lost. In particular, the exchange message is added to the knowledge of the attacker as soon as the agent emitting the message is a direct neighbour of a malicious node. This reflects the fact that a malicious node can listen to the communications of its neighbours since messages are typically broadcast to all neighbours. The In rule allows a malicious node to send any message it can deduce to one of its neighbours. The other rules are standard.

The relation $\to^*_{\mathcal{T}}$ is the reflexive and transitive closure of $\to_{\mathcal{T}}$.

Example 6. Continuing the example developed in Section 3.2, the following sequence of transitions is enabled from the initial configuration K_0.

$$K_0 \to^*_{\mathcal{T}_0} \big(\lfloor \mathsf{in}(u_2).\mathsf{if}\ \Phi_S\ \mathsf{then}\ 0\rfloor_S \cup \lfloor P_{\mathsf{dest}}(D)\rfloor_D ; \mathcal{I}_0 \cup \{u_1\}\big)$$

$$\text{where} \quad \begin{cases} u_1 = \langle req, S, D, id, S :: \bot, hmac(\langle req, S, D, id\rangle, shk(S, D))\rangle \\ u_2 = \langle rep, D, S, id, x_L, hmac(\langle rep, D, S, id, x_L\rangle, shk(S, D))\rangle \\ \Phi_S = \mathsf{checkl}(S, x_L) \wedge \mathsf{first}(D, x_L) \wedge \mathsf{last}(S, x_L) \end{cases}$$

During this transition, S broadcasts a request to find a route to D to its neighbours. The intruder I is a neighbour of S in $\mathcal{T}_0$, so he learns the request message. Assuming that the intruder knows the names of its neighbours, *i.e.* $A_1, A_2 \in \mathcal{I}_0$, he can then build the following fake message request:

$$m = \langle req, S, D, id, [A_1; A_2; S], hmac(\langle req, S, D, id\rangle, shk(S, D))\rangle$$

and send it to D. Indeed, we have that $\mathcal{I}_0 \vdash m$ (see Example 2).

Since $(A_1, D) \in E$, D accepts this message and the resulting configuration of the transition is $\big(\lfloor \mathsf{in}(u_2).\mathsf{if}\ \Phi_S\ \mathsf{then}\ 0\rfloor_S \cup \lfloor \mathsf{out}(v_2\sigma).0\rfloor_D ; \mathcal{I}_0 \cup \{u_1\}\big)$ where:

$$v_2 = \langle rep, D, S, x_{id}, D :: x_A :: x_l, hmac(\langle rep, D, S, x_{id}, D :: x_A :: x_l\rangle, shk(S, D))\rangle$$
$$\sigma = \{x_{id} \mapsto id, x_A \mapsto A_1, x_l \mapsto [A_2; S]\}.$$

3.5 Security properties

Routing protocols aim at establishing a valid route between two nodes S and D, that is a route that represents an existing path from S to D in the graph representing the network topology. However, it is well-known that the presence of several colluding malicious nodes often yields straightforward attacks, the so-called wormhole attacks (*e.g.* [22, 24]). Indeed, as soon as a malicious node is on the way of the request, he can behave as if he was a neighbour of another malicious node. This is a fact that our definition of security must tolerate, otherwise we cannot hope that any routing protocol will satisfy it. This observation leads to the following definition of *admissible path*.

Definition 1 (admissible path in $\mathcal{T}$). *Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology with $G = (V, E)$. We say that a list $l = [A_1, \ldots, A_n]$ of agent names is an* admissible path *in $\mathcal{T}$ if for any $i \in \{1, \ldots, i-1\}$, $(A_i, A_{i+1}) \notin E$ implies that $A_i \in \mathcal{M}$ and $A_{i+1} \in \mathcal{M}$.*

Another option could be to consider a weaker attacker model, assuming that the attackers can not communicate using an out-of-band channel, and to consider a stronger security property requiring the path to be a real path in G. In such a setting, routing protocols are often vulnerable to hidden channel attacks (see *e.g.* [14]). In our setting, this type of attack would not be considered as an attack, as it is an instantiation of the so-called wormhole attack that consists, for two dishonest nodes, in making the network believe they are neighbors.

After having successfully executed a routing protocol, the source node typically stored the resulting received route. For the sake of simplicity, we assume that processes representing instances of routing protocols contain a process (typically a session of the source node) that contains a special action of the form $\mathsf{out}(\mathit{end}(l))$. Checking whether a routing protocol ensures the validity of accepted routes can be defined as a reachability property.

Definition 2 (attack on a configuration K in $\mathcal{T}$). *Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology and K be a configuration. We say that K* admits an attack in $\mathcal{T}$ *if $K \rightarrow^*_{\mathcal{T}} (\lfloor \mathsf{out}(\mathit{end}(l)).P \rfloor_A \cup \mathcal{P}; \mathcal{I})$ for some $A, P, \mathcal{P}, \mathcal{I}$, and some term l that is not an admissible path in $\mathcal{T}$.*

Example 7. For the SRP protocol, we recover the attack mentioned in [15] with the topology given in Example 5, and from the initial configuration:

$$K_{\mathsf{init}} = \big(\lfloor P_0(S,D) \rfloor_S \mid \lfloor P_{\mathsf{dest}}(D) \rfloor_D ; \mathcal{I}_0\big)$$

where $P_0(x_S, x_D)$ is $P_{\mathsf{src}}(x_S, x_D)$ in which the null process 0 has been replaced by $\mathsf{out}(\mathit{end}(x_L)).0$.
Indeed, we have that:

$$\begin{aligned}
K_{\mathsf{init}} &\rightarrow^*_{\mathcal{T}_0} (\lfloor \mathsf{in}(u_2).\mathsf{if}\ \Phi_S\ \mathsf{then}\ P \rfloor_S \cup \lfloor \mathsf{out}(m').0 \rfloor_D ; \mathcal{I}) \\
&\rightarrow_{\mathcal{T}_0} (\lfloor \mathsf{in}(u_2).\mathsf{if}\ \Phi_S\ \mathsf{then}\ P \rfloor_S \cup \lfloor 0 \rfloor_D ; \mathcal{I}') \\
&\rightarrow_{\mathcal{T}_0} (\lfloor \mathsf{out}(\mathit{end}([D; A_1; A_2; S])).0 \rfloor_S ; \mathcal{I}')
\end{aligned}$$

where:

$m' = \langle \mathit{rep}, D, S, \mathit{id}, [D; A_1; A_2; S], \mathit{hmac}(\langle \mathit{rep}, D, S, \mathit{id}, [D; A_1; A_2; S] \rangle, \mathit{shk}(S, D)) \rangle$
$\mathcal{I} = \mathcal{I}_0 \cup \{u_1\}$ with $u_1 = \langle \mathit{req}, S, D, \mathit{id}, S :: \bot, \mathit{hmac}(\langle \mathit{req}, S, D, \mathit{id} \rangle, \mathit{shk}(S, D)) \rangle$
$\mathcal{I}' = \mathcal{I}_0 \cup \{u_1\} \cup \{m'\}$.

The list $[D; A_1; A_2; S]$ is not an admissible path in $\mathcal{T}_0$. Indeed, $(A_1, A_2) \notin E_0$ whereas A_1 and A_2 are both honest nodes, *i.e.* not in $\mathcal{M}$.

4 Reduction results

Our main contribution is a reduction result that allows one to analyse the security of a routing protocol considering only some specific and small topologies (typically the underlying graph will contain four nodes). Our reduction result is established in two main steps.

1. We show that the existence of an attack is preserved when adding edges to the graph, actually added all edges but one, yielding a *quasi-complete* topology (see Section 4.1).
2. We show how we can merge almost all nodes together, yielding a graph with only four nodes (see Section 4.2).

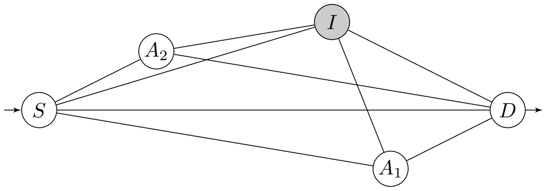
We finally conclude in Section 4.3 exhibiting five particular network topologies such that if there exists a network topology admitting an attack then there is an attack on one of the five exhibited topologies. This reduction result drastically reduces the search space (from infinitely many to only five network topologies). As a consequence, it is possible to analyse routing protocols using existing tools, *e.g.* the AVISPA platform [6] or the ProVerif tool [12], provided the protocols perform only actions supported by the tool.

4.1 From an arbitrary topology to a quasi-complete one

The main idea of our reduction result consists of projecting agents/nodes to the same node. However, we can only do that safely when the agents have the same status (honest/dishonest) and the same neighbourhood. The purpose of the first step (completing the graph) is to ensure that most of the nodes will have the same neighbourhood. This will ensure us to obtain a small graph after the merging step. Of course, we can not consider a complete graph since then any route would be valid thus there would not be any attack. The most complete topology on which an attack can be mounted is a *quasi-complete topology.*

Definition 3 (quasi-completion). *Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology with $G = (V, E)$, and A, B be two nodes in V that are not both in $\mathcal{M}$, and such that $(A, B) \notin E$. The quasi-completion of $\mathcal{T}$ w.r.t. (A, B) is a topology $\mathcal{T}^+ = (G^+, \mathcal{M}, S, D)$ such that $G^+ = (V, E^+)$ with $E^+ = V \times V \smallsetminus \{(A, B); (B, A)\}$.*

Example 8. The quasi-completion of the topology $\mathcal{T}_0 = (G_0, \mathcal{M}_0, S, D)$ (defined in Example 5) w.r.t. (A_1, A_2) is the topology $\mathcal{T}_0^+ = (G_0^+, \mathcal{M}_0, S, D)$ described below. The only missing edge is (A_1, A_2).



Note that the execution described in Example 7 is still an execution w.r.t. the topology $\mathcal{T}_0^+$ and this execution leads to an attack. This result holds for any protocol that uses completion-friendly predicates (see Proposition 1).

Definition 4 (completion-friendly). *A predicate p is* completion-friendly *if $[\![p(u_1, \dots, u_k)]\!]_G = \mathsf{true}$ implies $[\![p(u_1, \dots, u_k)]\!]_{G^+} = \mathsf{true}$ for any ground terms $u_1, \dots, u_k$, and quasi-completion $\mathcal{T}^+ = (G^+, \mathcal{M}, S, D)$ of $\mathcal{T} = (G, \mathcal{M}, S, D)$.*

We say that a configuration (resp. a routing protocol) is completion-friendly if $\mathcal{P}_D$ (i.e. the predicates that are dependent of the graph) are completion-friendly.

Example 9. All the predicates that do not depend on the underlying graph are completion-friendly. The predicates check and checkl (see Example 3) that are those used in our running example are completion-friendly whereas their negation are not. This allows us to conclude that the routing protocol $\mathcal{P}_{\mathsf{SRP}}$ is completion-friendly.

Proposition 1. *Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology and K be a configuration that is completion-friendly. If there is an attack on K in $\mathcal{T}$, then there exists an attack on K in $\mathcal{T}^+$ where $\mathcal{T}^+$ is a quasi-completion of $\mathcal{T}$. Moreover, $\mathcal{T}^+$ is a quasi-completion of $\mathcal{T}$ w.r.t. a pair (A_1, A_2) such that $A_1 \notin \mathcal{M}$ or $A_2 \notin \mathcal{M}$.*

4.2 Reducing the size of the topology

Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology where $G = (V, E)$ with E a reflexive relation, and ρ be a renaming on the agent names (not necessarily a bijective one). We say that the renaming ρ

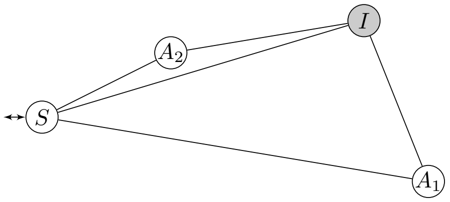
- *preserves neighbourhood of $\mathcal{T}$* if $\rho(A) = \rho(B)$ implies that
$$\{A' \in V \mid (A, A') \in E\} = \{B' \in V \mid (B, B') \in E\}$$
- *preserves honesty of $\mathcal{T}$* if $\rho(A) = \rho(B)$ implies that $A, B \in \mathcal{M}$ or $A, B \notin \mathcal{M}$.

Given a term u, we denote by $u\rho$ the term obtained by applying the renaming ρ on u. This notation is extended to set of terms, configurations, graphs, and topologies. In particular, given a graph $G = (V, E)$, we denote $G\rho$ the graph $(V\rho, E')$ such that $E' = \{(\rho(A), \rho(B)) \mid (A, B) \in E\}$.

Example 10. Going back to our running example, we may want to consider the identity renaming on agent names. Such a renaming preserves neighbourhood and honesty but it is not really interesting since it does not allow us to reduce the size of the topology, *i.e.* the number of vertices in the graph. A more interesting renaming that preserves neighbourhood and honesty of $\mathcal{T}_0^+$ is ρ defined as follows:

$$\rho(A_1) = A_1,\ \rho(A_2) = A_2,\ \rho(S) = \rho(D) = S, \text{ and } \rho(I) = I$$

Note that ρ does not preserve neighbourhood of the topology $\mathcal{T}_0$ (thus the completion step is important). The topology $\mathcal{T}_0^+\rho$ is described below:



In order to safely merge nodes together, we need the predicates and the functions to be stable over renaming of names of sort agent.

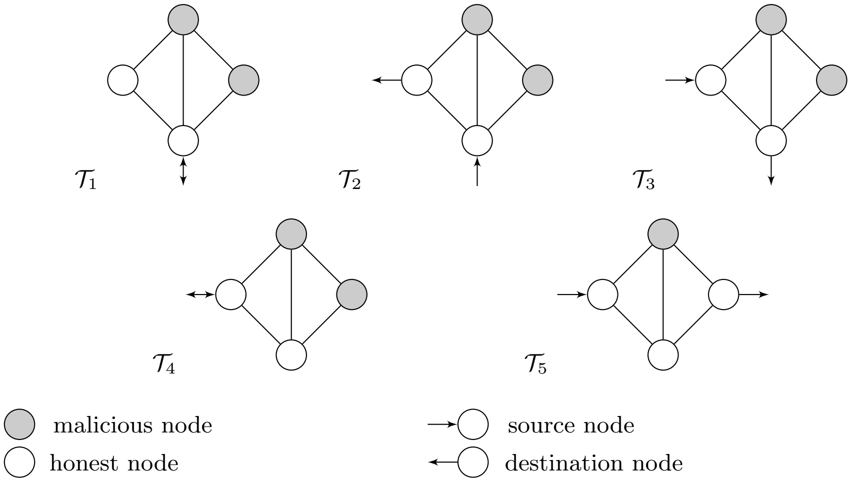


Fig. 3. Topologies $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$, $\mathcal{T}_4$, and $\mathcal{T}_5$.

Definition 5 (projection-friendly). *A predicate p is* projection-friendly *if $[\![p(u_1, \ldots, u_k)]\!]_G = \mathsf{true}$ implies $[\![p(u_1\rho, \ldots, u_k\rho)]\!]_{G\rho} = \mathsf{true}$ for any ground terms $u_1, \ldots, u_k$ and any renaming ρ that preserves neighbourhood and honesty.*

A function f over terms of arity k is projection-friendly *if $\mathsf{f}(u_1\rho, \ldots, u_k\rho) = \mathsf{f}(u_1, \ldots, u_k)\rho$ for any ground terms $u_1, \ldots, u_k$ and any renaming ρ that preserves neighbourhood and honesty.*

We say that a routing protocol (resp. a configuration) is projection-friendly *if the predicates in $\mathcal{P}_I \cup \mathcal{P}_D$, and the functions in $\mathcal{F}$ are projection-friendly.*

Example 11. The predicates check and checkl are projection-friendly since we consider renaming that preserves neighbourhood. In our running example, the set $\mathcal{F}_{\mathsf{SRP}}$ only contains the identity function. Clearly, this function is projection-preserving. More generally, all examples of functions provided in Section 2.3 are projection-friendly. Some predicates such as checking disequality constraints or verifying whether an agent name occurs twice in a list are not projection-friendly.

Proposition 2. *Let $\mathcal{T}$ be a topology, K_0 be a configuration that is projection-friendly, and ρ be a renaming that preserves neighbourhood and honesty of $\mathcal{T}$. If there is an attack on K_0 in $\mathcal{T}$, then there exists an attack on K'_0 in $\mathcal{T}'$ where K'_0 and $\mathcal{T}'$ are obtained by applying ρ on K_0 and $\mathcal{T}$.*

4.3 Only five topologies are sufficient!

Relying on Proposition 1 and Proposition 2, we are now able to show that the existence of an attack on a routing protocol that is completion-friendly and projection-friendly can be reduced to the problem of deciding the existence of an attack for the five topologies given in Figure 3.

Our reduction result is even slightly stronger as we show that we can actually also reduce the initial knowledge of the attackers, considering only the keys associated to the four nodes appearing in the topologies defined in Figure 3. Typical initial knowledges for the attacker are defined as the union of some public information from any agent and private information from malicious agents. More precisely, we assume that such a knowledge is given by a *template* $\mathcal{I}_0$, *i.e.* a set of terms in $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X}_V \cup \mathcal{X}_\mathcal{M})$ where $\mathcal{X}_V$ and $\mathcal{X}_\mathcal{M}$ are two disjoint sets of variables of sort agent where $\mathcal{X}_V$ represents all the nodes while $\mathcal{X}_\mathcal{M}$ represent the malicious nodes. Moreover, we assume that the only subterms of sort agent in $\mathcal{I}_0$ are the variables in $\mathcal{X}_V$ and $\mathcal{X}_\mathcal{M}$ Then, given a set of nodes V and a set of malicious nodes $\mathcal{M}$, the *knowledge* $\mathsf{Knowledge}(\mathcal{I}_0, V, \mathcal{M})$ *derived from the template* $\mathcal{I}_0$ is obtained by considering all possible substitutions that preserve the honesty status:

$$\mathsf{Knowledge}(\mathcal{I}_0, V, \mathcal{M}) = \left\{ (t\sigma_V)\sigma_\mathcal{M} \;\middle|\; \begin{array}{l} t \in \mathcal{I}_0, dom(\sigma_V) = \mathcal{X}_V, img(\sigma_V) \subseteq V, \\ dom(\sigma_\mathcal{M}) = \mathcal{X}_\mathcal{M}, \text{ and } img(\sigma_\mathcal{M}) \subseteq \mathcal{M} \end{array} \right\}$$

Example 12. For instance, this allows us to express that the attackers know all the public keys, and all the private keys that belong to malicious nodes:

$$\mathcal{I}_0 = \{x_v, pk(x_v), sk(x_m), shk(x_v, x_m), shk(x_m, x_v)\}$$

with $x_v \in \mathcal{X}_V$, and $x_m \in \mathcal{X}_\mathcal{M}$.

Definition 6 (configuration valid for $\mathcal{P}_\mathsf{routing}$ and P_0 w.r.t. $\mathcal{T}$ and $\mathcal{I}_0$). *Let $\mathcal{T} = (G, \mathcal{M}, S, D)$ be a topology where $G = (V, E)$, and $\mathcal{I}_0$ be a template representing the initial knowledge. A configuration $K = (\mathcal{P}, \mathcal{I})$ is valid for the routing protocol $\mathcal{P}_\mathsf{routing}$ and the routing role P_0 w.r.t. $\mathcal{T}$ and $\mathcal{I}_0$ if*

1. *$\mathcal{P} = \lfloor P_0(S, D) \rfloor_S \uplus \mathcal{P}'$ and for each $\lfloor P' \rfloor_{A_1} \in \mathcal{P}'$ there exists $P(x_1, \ldots, x_k) \in \mathcal{P}_\mathsf{routing}$, and $A_2, \ldots, A_k \in V$ such that $P' = P(A_1, \ldots, A_k)$; and*
2. *the only process containing a special action of the form $\mathsf{out}(end(l))$ is $P_0(S, D)$ witnessing the storage of a route by the source node S;*
3. *$\mathcal{I} = \mathsf{Knowledge}(\mathcal{I}_0, V, \mathcal{M})$.*

The first condition says that we only consider configurations that are made up using $P_0(S, D)$ and roles of the protocol, and the agent who executes the process is located at the right place. Moreover, we check whether the security property holds when the source and the destination are honest. Note that, we consider the case where an honest source initiates a session with a malicious nodes ($P_\mathsf{src}(S, I)$) can occur in the configuration). The second condition ensures that the process witnessing the route is the process $P_0(S, D)$.

Definition 7 (attack on $\mathcal{P}_\mathsf{routing}$ and P_0 w.r.t. $\mathcal{I}_0$). *We say that there is an attack on the routing protocol $\mathcal{P}_\mathsf{routing}$ and the routing role P_0 w.r.t. the template $\mathcal{I}_0$ if there exists a topology $\mathcal{T} = (G, \mathcal{M}, S, D)$ and a configuration K that is valid for $\mathcal{P}_\mathsf{routing}$ and P_0 w.r.t. $\mathcal{T}$ and $\mathcal{I}_0$ such that K admits an attack in $\mathcal{T}$.*

If there is an attack, then there is an attack on one of the five topologies depicted in Figure 3.

Theorem 1. *Let $\mathcal{P}_{\mathsf{routing}}$ be a routing protocol and P_0 be a routing role that are both completion-friendly and projection-friendly and $\mathcal{I}_0$ be a template.*

There is an attack on $\mathcal{P}_{\mathsf{routing}}$ and P_0 w.r.t. $\mathcal{I}_0$ for some topology $\mathcal{T}$ if, and only if, there is an attack on $\mathcal{P}_{\mathsf{routing}}$ and P_0 w.r.t. $\mathcal{I}_0$ for one of the topologies depicted in Figure 3.

The proof of Theorem 1 follows from the successive applications of graph completion (Proposition 1) and nodes projection (Proposition 2). Our result holds for an unbounded number of sessions since we consider arbitrarily many instances of the roles occurring in $\mathcal{P}_{\mathsf{routing}}$, and it encompasses many families of routing protocols.

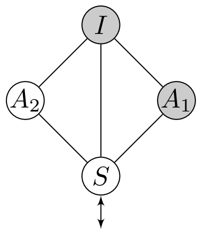
Corollary 1. *Let $\mathcal{P}_{\mathsf{routing}}$ be a routing protocol and P_0 be a routing role that are both built using functions over terms defined in Section 2.3 and predicates defined in Example 3, and $\mathcal{I}_0$ be a template.*

There is an attack on $\mathcal{P}_{\mathsf{routing}}$ and P_0 w.r.t. $\mathcal{I}_0$ for some topology $\mathcal{T}$ if, and only if, there is an attack on $\mathcal{P}_{\mathsf{routing}}$ and P_0 w.r.t. $\mathcal{I}_0$ for one of the topologies depicted in Figure 3.

Interestingly, a more careful analysis of the proof shows that our reduction result strictly preserves the number of sessions: if there is an attack on an arbitrary topology with k sessions, then there is an attack with k sessions for one of the topologies of Figure 3. Therefore our result holds for a bounded number of sessions as well.

Example 13. Going back to our running example, the topology $\mathcal{T}_0$ on which an attack has been found does not correspond to one of the topologies presented in Figure 3. However, we can retrieve the attack by considering the topology $\mathcal{T}_1$.

The attack described in Example 7 is obtained considering the template $\mathcal{I}_0 = \{x_V\}$ with $x_V \in \mathcal{X}_V$ which corresponds to an attacker who knows the names of all the agents. The topology $\mathcal{T}_1$ (see the picture on the right) does not correspond exactly to the topology $\mathcal{T}_0^+\rho$, *i.e.* the one obtained after completion and projection of $\mathcal{T}_0$. Indeed, the node A_1 is assumed to be malicious in the topology $\mathcal{T}_1$, but not in $\mathcal{T}_0^+\rho$. Note that the attack still exists in presence of this additional malicious node.



We consider the configuration $K'_{\mathsf{init}} = (\lfloor P_0(S,S)\rfloor_S \mid \lfloor P_{\mathsf{dest}}(S)\rfloor_S; \mathcal{I}')$ where $\mathcal{I}'_0 = \{A_1; A_2; I; S\}$. Since S is an honest node, this configuration is a valid configuration w.r.t. $\mathcal{T}$ and $\mathcal{I}_0$. We have that:

$$\begin{aligned} K'_{\mathsf{init}} &\rightarrow^*_{\mathcal{T}_1} (\lfloor \mathsf{in}(u_2).\mathsf{if}\ \Phi_S\ \mathsf{then}\ P\rfloor_S \cup \lfloor \mathsf{out}(m').0\rfloor_S; \mathcal{I}) \\ &\rightarrow_{\mathcal{T}_1} (\lfloor \mathsf{in}(u_2).\mathsf{if}\ \Phi_S\ \mathsf{then}\ P\rfloor_S \cup \lfloor 0\rfloor_S; \mathcal{I}') \\ &\rightarrow_{\mathcal{T}_1} (\lfloor \mathsf{out}(end([S; A_1; A_2; S])).0\rfloor_S; \mathcal{I}') \end{aligned}$$

where $m' = \langle rep, S, S, id, l_{route}, hmac(\langle rep, S, S, id, l_{route}\rangle, shk(S,S))\rangle$ and $l_{route} = [S; A_1; A_2; S]$. The list $[S; A_1; A_2; S]$ is not an admissible path in $\mathcal{T}_1$.

5 Case studies using ProVerif

In this section, relying on our reduction result, we propose an analysis of the SRP applied to DSR and the SDMSR protocols using the ProVerif tool [12].

5.1 Proverif

ProVerif constitutes a well-established automated protocol verifier based on Horn clauses resolution that allows for the verification of several security properties. The tool takes as input processes written in a syntax close to the one described in Section 3. It does not consider arbitrary functions over terms as we did, but it can handle many different cryptographic primitives, including shared and public key cryptography (encryption and signatures), hash functions, lists, ... It can handle an unbounded number of sessions of the protocol and an unbounded message space. This is possible thanks to some well-chosen approximations, which means that the tool can give false attacks. Actually, the tool may return three kinds of answer: either an attack is found (and ProVerif gives the attack trace), or no attack is found (but this does not mean that the protocol is secure), or else the protocol is proved secure.

It is interesting to notice that for the five topologies we have characterized, we can safely consider an attacker who listens all the communication channels. Moreover, we can easily encode neighbourhood checks, or the property to be an admissible path by defining predicates through Horn clauses. For instance, the predicate $\mathsf{check}(A, B)$ can be defined by enumerating all the existing links in the four-nodes topology under study.

5.2 Case studies

As an application, we consider two source routing protocols. The first one is the protocol SRP applied on DSR that has been described in Section 3.2. We also studied the multipath routing protocol SDMSR introduced in [11] whose aim is to find several paths leading from the source S to the destination D.

We give below a brief description of the SDMSR protocol (a more detailed description can be found in [17]). First, the source constructs a request packet and broadcasts it to its neighbours.

$$\langle req, S, D, id, S :: [], [\![\langle req, S, D, id\rangle]\!]_{sk(S)}\rangle.$$

This packet contains in particular the beginning of a route to D, and a signature over the content of the request, computed with the private key $sk(S)$. The source then waits for a reply containing a route to D signed by one of his neighbours, and checks that this route is plausible. The names of the intermediate nodes are accumulated in the route request packet and the attached signature is checked by each intermediate node. When the request reaches the destination D via the node B, he performs some checks and constructs a route reply.

$$\langle rep, D, x_S, id, D, l_{route}, [\![\langle rep, D, S, id, l_{route}\rangle]\!]_{sk(D)}\rangle.$$

In particular it computes a signature over the route l_{route} accumulated in the request packet with its private key $sk(D)$. It then sends the reply back over the network. The reply travels along the route back to S, and the intermediate nodes check that the signature in the reply packet is correct, and that the route is plausible, before forwarding it. Each node V replaces the signature $[\![\langle rep, D, S, id, l_{route}\rangle]\!]_{sk(A)}$ computed by its neighbour by its own signature $[\![\langle rep, D, S, id, l_{route}\rangle]\!]_{sk(V)}$.

5.3 Results

We analyse these protocols considering the five different topologies that have been described in Section 4.3 and for an unbounded number of sessions. We analyse the configuration where each node of the topology plays an unbounded number of sessions of each role (each node can act as a source, a destination, or an intermediate node).

Note also that even if ProVerif is able to manipulate lists and predicates defined through Horn clauses, those predicates are quite powerful and ProVerif is not always able to handle them in a satisfactory way. Therefore, we did not model the sanity check $\mathsf{last}(S, x_L)$ that is normally performed by the source. But this did not introduce any false attack: the attacks that are reported by the ProVerif tool are still valid when considering this additional check.

	SRP applied on DSR	SDMSR
$\mathcal{T}_1$	**attack found**	**attack found**
$\mathcal{T}_2$	**attack found**	**attack found**
$\mathcal{T}_3$	no attack found	no attack found
$\mathcal{T}_4$	no attack found	no attack found
$\mathcal{T}_5$	no attack found	no attack found

We retrieve the attack on the protocol SRP applied to DSR, mentioned in Example 7. Actually, the SDMSR protocol is subject to the same kind of attack than SRP applied to DSR (see [9]). The running time of ProVerif was less than a few secondes for each topology. All the files for these experiments are available at: http://www.lsv.ens-cachan.fr/~delaune/RoutingProtocols.

6 Conclusion

When checking whether a routing protocol ensures that resulting routes are valid even in the presence of malicious nodes, we have shown a simple reduction result: there is an attack on an arbitrary topology if and only if there is an attack on one of five particular topologies, each of them having only four nodes. It is therefore possible to use standard verification tools for analysing routing protocols, provided they make use of primitives supported by the tools.

Our execution model of protocols is very general, encompassing many families of routing protocols, *e.g.* with recursive tests/operations and various kinds of neighbourhood checks. Our only restriction is the fact that tests should be stable under projection of nodes names, typically disallowing test of difference. Disequality tests are useful to discard a route that contains twice the same node, or for checking disequality of session ids to avoid answering twice the same request. Investigating how to extend our reduction result to some families of difference tests is left for future work. Also, the five topologies we obtain may seem unrealistic, for example because the source and the destination are neighbours. It seems feasible to refine our reduction result adding some topological constraints such as avoiding the source and the destination to be neighbours, possibly considering a larger (but still finite) number of nodes. A limitation of our work is the fact that it is limited to a single (crucial) property: the validity of the resulting route. Our reduction result certainly works for other properties. But understanding (and formalizing) which security properties are relevant for routing protocols is a difficult question. Another extension would be to model mobility during the execution of the protocol. This would allow us to consider changes in the network topology and to analyze the security of route updates. This requires to model an appropriate security property.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.
2. M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th Conference on Computer and Communications Security (CCS'97)*, pages 36–47. ACM Press, 1997.
3. G. Ács, L. Buttyán, and I. Vajda. Provably secure on-demand source routing in mobile ad hoc networks. *IEEE Trans. Mob. Comput.*, 5(11):1533–1546, 2006.
4. T. Andel, G. Back, and A. Yasinsac. Automating the security analysis process of secure ad hoc routing protocols. *Simulation Modelling Practice and Theory*, 19(9):2032 – 2049, 2011.
5. A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In *Proc. of the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)*, pages 1–10. ACM, 2008.
6. A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Proc. 17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.
7. M. Arnaud, V. Cortier, and S. Delaune. Modeling and verifying ad hoc routing protocols. In *Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 59–74. IEEE Computer Society Press, July 2010.
8. M. Arnaud, V. Cortier, and S. Delaune. Deciding security for protocols with recursive tests. In *Proc. 23rd International Conference on Automated Deduction (CADE'11)*, LNAI, pages 49–63, Wrocław, Poland, 2011. Springer.
9. M. Arnaud, V. Cortier, and S. Delaune. Modeling and verifying ad hoc routing protocols. Research Report LSV-11-24, Laboratoire Spécification et Vérification, ENS Cachan, France, Dec. 2011. 68 pages.

10. D. Benetti, M. Merro, and L. Viganò. Model checking ad hoc network routing protocols: Aran vs. endaira. In *Proc. 8th IEEE International Conference on Software Engineering and Formal Methods, (SEFM'10)*, pages 191–202, Pisa, Italy, 2010. IEEE Computer Society.
11. S. Berton, H. Yin, C. Lin, and G. Min. Secure, disjoint, multipath source routing protocol(sdmsr) for mobile ad-hoc networks. In *Proc. 5th International Conference on Grid and Cooperative Computing*, GCC '06, pages 387–394, Washington, DC, USA, 2006. IEEE Computer Society.
12. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Comp. Soc. Press, 2001.
13. B. Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *Proc. 20th International Conference on Automated Deduction (CADE'05)*, 2005.
14. M. Burmester and B. de Medeiros. On the security of route discovery in manets. *IEEE Trans. Mob. Comput.*, 8(9):1180–1188, 2009.
15. L. Buttyán and I. Vajda. Towards Provable Security for Ad Hoc Routing Protocols. In *Proc. 2nd ACM workshop on Security of ad hoc and sensor networks (SASN'04)*, pages 94–105, New York, NY, USA, 2004. ACM.
16. H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *Proc. 12th European Symposium On Programming (ESOP'03)*, volume 2618 of *LNCS*, pages 99–113. Springer Verlag, April 2003.
17. V. Cortier, J. Degrieck, and S. Delaune. Analysing routing protocols: four nodes topologies are sufficient. Research Report LSV-11-25, Laboratoire Spécification et Vérification, ENS Cachan, France, Dec. 2011. 28 pages.
18. C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proc. 20th International Conference on Computer Aided Verification (CAV 2008)*, volume 5123/2008 of *LNCS*, pages 414–418. Springer, 2008.
19. D. Dolev and A. C. Yao. On the security of public key protocols. In *Proc. 22nd Symposium on Foundations of Computer Science (FCS'81)*, pages 350–357. IEEE Computer Society Press, 1981.
20. T. Feng, X. Guo, J. Ma, and X. Li. UC-Secure Source Routing Protocol, 2009.
21. Y.-C. Hu, A. Perrig, and D. Johnson. Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks. *Wireless Networks*, 11:21–38, 2005.
22. Y.-C. Hu, A. Perrig, and D. B. Johnson. Wormhole attacks in wireless networks. *Selected Areas in Communications, IEEE Journal on*, 24(2):370–380, 2006.
23. D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. In *In Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5*, pages 139–172. Addison-Wesley, 2001.
24. L. Lazos, R. Poovendran, C. Meadows, P. Syverson, and L. W. Chang. Preventing wormhole attacks on wireless ad hoc networks: a graph theoretic approach. In *Wireless Communications and Networking Conference*, volume 2, 2005.
25. S. Nanz and C. Hankin. A Framework for Security Analysis of Mobile Wireless Networks. *Theoretical Computer Science*, 367(1):203–227, 2006.
26. P. Papadimitratos and Z. Haas. Secure routing for mobile ad hoc networks. In *Proc. SCS Communication Networks and Distributed Systems Modelling Simulation Conference (CNDS)*, 2002.
27. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190. IEEE Comp. Soc. Press, 2001.