Formal Verification of the ARAN Protocol Using the Applied π-Calculus

Jens Chr. Godskesen [*]

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S, Denmark
`jcg@itu.dk`

Abstract. In this paper we carry out an analysis of the ARAN protocol for secure routing in ad hoc networks. The protocol is modeled in the Applied π-Calculus and the analysis is carried out using the verification tool ProVerif. We conjecture that the protocol, despite its use of certificates and signing of messages, is erroneous. We propose an addendum to the protocol that we claim to be a guard against the identified attack.

1 Introduction

The use of wireless networks is becoming more and more important due to the increasing and widespread use of communicating mobile devices. Securing these kinds of networks against missuses and attacks has therefore been the subject of much research recently. In this paper we study *mobile ad hoc networks* which are self organizing networks without centralized access points and a pre-deployed infrastructure for routing messages. An ad hoc network may be formed when a collection of mobile nodes join together and agree on how to route messages for each other.

Most *routing protocols* for ad hoc networks, say the reactive protocols DSR [9] and AODV [16], or the proactive protocols DSDV [15] and OLSR [17] are focusing on providing routing services without considering security and hence inherently all participants are assumed to be trusted. This may of course be unrealistic in many environments, say a hostile environment like e.g. a military battle field, or other information sensitive environments like a police or a safety-critical business network.

Typical security problems caused by malicious nodes in an ad hoc network are: redirection of network traffic (including denial-of-service attacks and tunneling), attacks where the adversary impersonates another node (spoofing), and attacks using fabrication of false routing messages.

In order to alleviate these kind of problems recently *secure* routing protocols for ad hoc networks have been proposed, for instance Secure AODV [20], SEAD [8] (which is based on DSDV), SRP [14], ARIADNE [7], and ARAN [18].

[*] Supported by grant no. 272-05-0258 from the Danish Research Agency.

A lot of research has been devoted to automatic verification of cryptographic protocols for ordinary fixed networks, probably most notably is the work done by Lowe [10]. However, little emphasis has so far been put into showing correctness of secure routing protocols for ad hoc networks. One exception although is carried out in [11] where a *manual* proof of a flaw in SRP is established, but *automatic* proofs for ad hoc network protocols (e.g. [5, 3, 19]) are to the best of our knowledge only carried out for protocols not dealing with security. The novelty of this paper is that we focus on automatic verification of secure routing protocols for ad hoc networks.

In this paper we analyze the ARAN protocol by modeling it in the Applied π-Calculus [2] and using the tool ProVerif [4]. We show that false routes can be constructed by an adversary establishing a man-in-the-middle spoofing attack. Finally, we remedy the problem proposing a solution of how to solve such kind of attacks in ARAN.

2 The ARAN protocol

In the following we briefly describe the protocol being the subject of our studies. The name of the protocol is *Authenticated Routing for Ad hoc Networks* (ARAN) [18] and it is a proposal for a secure protocol for on-demand routing in ad hoc networks.

The novelty of ARAN compared to the standard insecure routing protocols is to make use of cryptographic certificates in each communication step between participating nodes. Therefore the preliminary step of ARAN is a certification process relying on a trusted certificate server T.

In this initial process it is assumed that all nodes know the public key, ${K_T}^+$, of the trusted server T, and also it is assumed that any node in the network, A, has a private and public key pair, ${K_A}^-$ and ${K_A}^+$ respectively. The preliminary process essentially means that before entering the network each node must securely authenticate its identity to T, and by so doing obtain a certificate. A certificate, $cert_A$, for a node A is defined by:

$$cert_A = [ip_A, {K_A}^+, t, e]_{{K_T}^-} \quad ,$$

where ip_A is the IP address of A, ${K_A}^+$ its public key, and t and e are timestamps for the creation and expiration of the certificate respectively. The concatenation of the elements is digitally signed by T. [1]

2.1 Route discovery

After the preliminary certification step the validated nodes may begin their authenticated route discoveries. A source node A begins the instantiation of a

[1] The notation $[m]_K$ denotes both the message m and the signature of m generated by the key K.

route to a destination D by broadcasting a signed *route discovery message* (or package) to its neighbors:

$$A \to brdcast : [rdp, ip_D, n_A]_{K_A^-}, cert_A \quad ,$$

where rdp denotes the type of the message, ip_D the destination IP address, and n_A is a nonce; all signed by A. A's certificate is appended to the signed part of the message.

A node B receiving A's request remembers the initiator and sets up a reverse path back to A. Then B validates that A's certificate has not expired and extracts the public key from it in order to validate the signature of the request. B also checks, based on ip_A and n_A that it has not already processed the request previously. [2] If all checks are passed B signs the first part of the request, appends its own certificate, and broadcast a *forwarded request message*:

$$B \to brdcast : [[rdp, ip_D, n_A]_{K_A^-}]_{K_B^-}, cert_A, cert_B \quad .$$

A neighbor C receiving B's forwarded message validates A's and B's certificates and signatures, checks that it hasn't seen A's request before, records B as its predecessor in the back path, replaces B's certificate with its own, and broadcasts the request originally broadcasted by A but now as a forwarded request message signed by itself:

$$C \to brdcast : [[rdp, ip_D, n_A]_{K_A^-}]_{K_C^-}, cert_A, cert_C \quad .$$

Each node along the path to the destination repeats the step described above.

If the destination D receives a (forwarded) request messages from some node E it validates the certificates and the signatures as described above, it also checks that it has not answered the request before. [3]

D then unicasts a signed *reply messages* along the reverse path back to E:

$$D \to E : [rep, ip_A, n_A]_{K_D^-}, cert_D \quad ,$$

where rep is the type of the message, ip_A is the IP address of the request initiator A, and n_A is the nonce being part of the request by A, and $cert_D$ is D's certificate,

E validates the certificate and the signature of D's reply, signs the reply, and unicasts the reply together with its certificate as a *forwarded reply message* to its predecessor F in the back path:

$$E \to F : [[rep, ip_A, n_A]_{K_D^-}]_{K_E^-}, cert_D, cert_E \quad .$$

Following the pattern for the request messages, each node along the path back to the source validates the (forwarded) reply message, records the sender as the

[2] It is assumed that a nonce never reappears within the lifetime of the network so a nonce together with A's IP address uniquely identifies the request.

[3] Note that there is no guarantee that the shortest path from the source to the destination is then selected.

forwarding node towards D, removes the senders certificate and signature, signs the reply message itself, and adds its own certificate before forwarding the reply message to its predecessor.

In order to avoid replay attacks each node along the back path checks the nonce of the reply message. In addition to the above mentioned, ARAN also contains features for route maintenance and key revocation. We shall not further address these topics in this paper and for a detailed presentation we refer the reader to [18].

3 The Applied π-Calculus

The *Applied π-Calculus* [2] is a simple extension of the π-Calculus [13] with value passing, primitive functions, and term equations. Hence in contrast to the π-Calculus where only names are passed in the Applied π-Calculus messages may also consist of values constructed from names and functions. The modeling of security protocols has been a major motivating example during the development of the Applied π-Calculus.

Since a complete presentation of the Applied π-Calculus is outside the scope of this paper we restrict the description below to the sub-calculus that has been applied in our modeling of the ARAN protocol. The presentation is based on the variant of the calculus given in [1], the same variant that is implemented in the ProVerif tool [4] that is presented in Section 5.

3.1 Syntax

The syntax of the calculus is composed of *terms* (data) and *processes*. Terms are defined relative to an infinite set of *names*, an infinite set of *variables*, and two disjoint finite sets of *constructor* and *destructor* symbols. Each constructor and destructor symbol is equipped with an arity.

As an example, let $\{ok, pk, sk, sign\}$ be a set of constructor symbols and let $\{check, get\}$ be a set of destructor symbols where ok has arity 0 (i.e. a constant), where get, pk, and sk have arity 1, where $sign$ has arity 2, and where $check$ has arity 3. Constructors are used to build terms so ok is a term, and if s and t are two names then

$$sign(pk(s), sk(t)) \quad , \tag{1}$$

constitute a term. We may let $pk(s)$ be the constructor for a public key based on some seed s and we may let $sk(t)$ be a private (secret) key based on the seed t. The application of the constructor $sign$ then denotes the signing of the public key $pk(s)$ with the secret key $sk(t)$.

Destructor symbols do not appear in terms, instead a destructor based on the arguments it is given may produce a new term. How destructors precisely manipulate terms are defined by equations, each destructor is equipped with a finite set of defining equations. For instance, we may let the destructors $check$ and get be defined by:

$$check(M, sign(M, sk(N)), pk(N)) = ok \quad , \quad get(sign(M, sk(N))) = M \quad . \tag{2}$$

That is, checking the signature of a message M with the public key matching the private key by which the message was signed yields the result *ok*. The destructor *get* simply returns the contents of a signed message.

Formally, destructors are defined to be partial functions, i.e. the application of a destructor to a tuple of terms is only defined in case the tuple matches one of the destructors defining equations (we refer the interested reader to [1]).

Given constructor and destructor symbols, an infinite set of names, and an infinite set of variables the set of terms is defined as follows:

$M, N ::=$		terms
	a, b, c	names
	x, y, z	variables
	$f(M_1, \ldots, M_k)$	constructor application

where f is a constructor symbol with arity k.

The processes in the Applied π-Calculus are mostly standard process constructs and naturally most of them are taken from the π-Calculus. Based on the set of terms defined above, including the infinite set of names, and the infinite set of variables, the set of all processes is defined by the grammar:

$P, Q ::=$		processes
	0	inactive process
	$P \parallel Q$	parallel composition
	$\nu a.P$	name restriction
	$\mathit{if}\ M = N\ \ \mathit{then}\ P\ \mathit{else}\ Q$	conditional
	$a(x_1, \ldots, x_k).P$	message input
	$\overline{a}\langle M_1, \ldots, M_k\rangle.P$	message output
	$\mathit{let}\ x = M\ \mathit{in}\ P$	local definition
	$\mathit{let}\ x = g(M_1, \ldots, M_k)\ \mathit{in}\ P \mathit{else}\ Q$	destructor application

The process 0 is the inactive process and $P \parallel Q$ is the parallel composition of P and Q. The process $\nu a.P$ binds the name a in P and restricts a to P. $\mathit{if}\ M = N\ \ \mathit{then}\ P\ \mathit{else}\ Q$ is a standard conditional. The process $a(x_1, \ldots, x_k).P$ binds $x_1, \ldots, x_k$ in P and may input terms $N_1, \ldots, N_k$ on channel a and in so doing replace all free occurrences of x_i in P by N_i, $i = 1, \ldots, k$. $\overline{a}\langle M_1, \ldots, M_k\rangle.P$ may output $M_1, \ldots, M_k$ on channel a and become P. The local definition $\mathit{let}\ x = M\ \ \mathit{in}\ P$ binds the variable x in P and executes P with all free occurrences of x replaced by M. The process $\mathit{let}\ x = g(M_1, \ldots, M_k)\ \mathit{in}\ P \mathit{else}\ Q$ also binds x in P, if the destructor application $g(M_1, \ldots, M_k)$ evaluates to a term then x is bound to the result in P, otherwise the process becomes Q. [4]

We let $P\{M_1/x_1, \ldots, M_k/x_k\}$ denote the process P where $x_1, \ldots, x_k$ are substituted by $M_1, \ldots, M_k$ respectively. The set of *free names* in a process P is denoted by $\mathit{fn}(P)$, and its *free variables* are denoted by $\mathit{fv}(P)$. A process P is *closed* if $\mathit{fv}(P) = \emptyset$. $\mathcal{P}$ denotes the set of all closed processes. As usual we identify processes up to α-equivalence.

[4] By convenience we have chosen a polyadic variant of the Applied π-Calculus, and we have left out replication because it is not used in the present paper.

As shorthands, whenever Q is 0, we abbreviate $\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q$ by $\mathit{if}\ M = N\ \mathit{then}\ P$, we write $\mathit{let}\ x = g(M_1, \ldots, M_k)\ \mathit{in}\ P$ instead of $\mathit{let}\ x = g(M_1, \ldots, M_k)\ \mathit{in}\ P\ \mathit{else}\ Q$, and also, we write $\overline{a}\langle M_1, \ldots, M_k \rangle$ for $\overline{a}\langle M_1, \ldots, M_k \rangle.Q$.

3.2 Semantics

The operational semantics of processes in the Applied π-Calculus is standardly given by a *reduction relation*, $\longrightarrow\ \subseteq \mathcal{P} \times \mathcal{P}$, defined with respect to a *structural congruence* denoted by $\equiv$. Formally, $\equiv\ \subseteq \mathcal{P} \times \mathcal{P}$ is the least equivalence relation that is also closed under parallel composition and name restriction, where $(\mathcal{P}, \|, 0)$ is a commutative monoid, and such that $\nu a.\nu b.P \equiv \nu b.\nu a.P$ and $\nu a.(P \parallel Q) \equiv P \parallel \nu a.Q$ if $a \notin fn(P)$.

The reduction relation $\longrightarrow\ \subseteq \mathcal{P} \times \mathcal{P}$ is the least relation closed under structural congruence, parallel composition, and name restriction, and also satisfying the rules in Table 1. [5]

$\overline{a}\langle M_1, \ldots, M_k \rangle.P \parallel a(x_1, \ldots, x_k).Q \longrightarrow P \parallel Q\{M_1/x, \ldots, M_k/x_k\}$	
$\mathit{let}\ x = g(M_1, \ldots, M_k)\ \mathit{in}\ P\ \mathit{else}\ Q \longrightarrow P\{M/x\}$,	if $g(M_1, \ldots, M_k) = M$
$\mathit{let}\ x = g(M_1, \ldots, M_k)\ \mathit{in}\ P\ \mathit{else}\ Q \longrightarrow Q$,	if $g(M_1, \ldots, M_k)$ is undefined
$\mathit{let}\ x = M\ \mathit{in}\ P \longrightarrow P\{M/x\}$	
$\mathit{if}\ M = N\ \ \mathit{then}\ P\ \mathit{else}\ Q \longrightarrow P$,	if $M = N$
$\mathit{if}\ M = N\ \ \mathit{then}\ P\ \mathit{else}\ Q \longrightarrow Q$,	if $M \neq N$

Table 1. Reduction Rules

To give an example, recall the constructors *sign*, *pk*, and *sk* and the destructor *get* from Section 3.1 above and let P and Q be defined by:

$$P \stackrel{\text{def}}{=} \mathit{let}\ x = \mathit{sign}(t, \mathit{sk}(s))\ \mathit{in}\ \overline{a}\langle \mathit{pk}(s), x \rangle \tag{3}$$

$$Q \stackrel{\text{def}}{=} a(x_1, x_2).\mathit{let}\ y = \mathit{get}(x_2)\ \mathit{in}\ \overline{a}\langle x_1, y \rangle \tag{4}$$

Then $P \longrightarrow \overline{a}\langle \mathit{pk}(s), \mathit{sign}(t, \mathit{sk}(s)) \rangle$ and

$$\begin{aligned} P \parallel Q &\longrightarrow \overline{a}\langle \mathit{pk}(s), \mathit{sign}(t, \mathit{sk}(s)) \rangle \parallel Q \\ &\longrightarrow \mathit{let}\ y = \mathit{get}(\mathit{sign}(t, \mathit{sk}(s)))\ \mathit{in}\ \overline{a}\langle \mathit{pk}(s), y \rangle \longrightarrow \overline{a}\langle \mathit{pk}(s), t \rangle \end{aligned} \tag{5}$$

[5] Notice that we have overloaded the use of the symbol = to both being part of the syntax of process expressions and also denoting equality in the equational theory on terms.

We say that a process P *outputs* M on channel a if there exists a sequence of transitions $P \longrightarrow^* \nu a_1 \ldots \nu a_l.(\overline{a}\langle M_1, \ldots, M_k\rangle.P_1 \parallel P_2)$, $a \notin \{a_1, \ldots, a_l\}$, and $M \in \{M_1, \ldots, M_k\}$. [6] Hence, the process P defined by (3) above outputs on channel a the public key $pk(s)$ and the message $sign(t, sk(s))$ signed by the corresponding secret key $sk(s)$. The parallel composition $P \parallel Q$ in the example (5) outputs the same as P on channel a as well as t and the public key $pk(s)$.

3.3 Secrecy

The authors of [1] define a notion of *secrecy* of data that turns out to be sufficient for our purposes. They define secrecy relative to a finite set of names S, the names (including channel names) assumed to be known in advance by an adversary. They let Q be an S-adversary if $fn(Q) \subseteq S$ and proceed defining:

Definition 1. *P preserves the secrecy of M from S if and only if $P \parallel Q$ does not output M on channel a for any S-adversary Q and any $a \in S$.*

As an example, the process P defined by (3) does not preserve the secrecy of $pk(s)$ and $sign(t, sk(s))$ from $\{a\}$ because it may send out the terms on a. Also, P does not preserve the secrecy of the name t from $\{a\}$ because sending out the composed term $sign(t, sk(s))$ may permit an $\{a\}$-adversary to compute t and send it out on a as demonstrated by (5) above choosing Q defined by (4) as the adversary. P does however preserve the secrecy of the seed s from any S-adversary where $s \notin S$ because although s is part of the term $sign(t, sk(s))$ sent out on a no such S-adversary may compute s (modulo the computational theory of our running example).

4 The ARAN model

In this section we provide a simplified model of the ARAN protocol as a process in the Applied π-Calculus. One simplification being that we do not explicitly model the preliminary phase of ARAN where certificates are distributed to nodes, instead we assume all valid nodes a priori to be the owner of a certificate issued by the trusted certificate server.

Another simplification we make is to assume that the validity of a certificate is not limited to a certain time interval, so we abstract from dealing with timestamps in certificates. Also, as described in Section 2, in the checking of recycling of messages the IP address of the owner of a certificate is an important ingredient, however since we are not going to deal with recycling attacks in this paper we choose to eliminate the IP address from a certificate. Hence we let a certificate be only a signed public key as defined by (1) in Section 3.1. The model makes use of two destructors, *check* and *get*, for dealing with certificates. Their defining equations are given by (2) in Section 3.1.

The third and final restriction is that our simplified model of the ARAN protocol is just a simple one-shot version of the protocol where only one request

[6] $\longrightarrow^*$ is the transitive and reflexive closure of $\longrightarrow$.

and its reply are carried out, and to demonstrate the attack in Section 6 it turns out to be sufficient to let only three nodes participate in the protocol, i.e. an initiator A_0 and a destination C_0 with an intermediary node B_0 that forwards request and reply messages.

Formally we let the ARAN model be defined by:

$$Aran \stackrel{\text{def}}{=} \nu t.(A_0 \mid B_0 \mid C_0) \quad , \tag{6}$$

where the name t, being private to the protocol, is assumed to be the seed for the keys of the trusted server issuing certificates.

$$\boxed{\begin{array}{l}
A_0 \stackrel{\text{def}}{=} \nu s.let\ x_{cert} = sign(pk(s), sk(t))\ in\ A_1 \\
A_1 \stackrel{\text{def}}{=} let\ x_{sreq} = sign(rdp, sk(s))\ in\ \overline{a_1}\langle x_{sreq}, x_{cert}\rangle.a_2(x_1, x_2, x_3).A_2 \\
A_2 \stackrel{\text{def}}{=} let\ x_{srep} = get(x_1)\ in\ if\ rep = get(x_{srep})\ then\ A_3 \\
A_3 \stackrel{\text{def}}{=} let\ x_{dkey} = get(x_2)\ in\ if\ check(x_{dkey}, x_2, pk(t)) = ok\ \ then\ A_4 \\
A_4 \stackrel{\text{def}}{=} if\ check(rep, x_{srep}, x_{dkey}) = ok\ \ then\ A_5 \\
A_5 \stackrel{\text{def}}{=} let\ x_{fkey} = get(x_3)\ in\ if\ check(x_{fkey}, x_3, pk(t)) = ok\ \ then\ A_6 \\
A_6 \stackrel{\text{def}}{=} if\ check(x_{srep}, x_1, x_{fkey}) = ok\ \ then\ \overline{a_3}\langle success\rangle
\end{array}}$$

Fig. 1. The request initiator process A_0.

The route request initiator A_0 is defined by the equations in Figure 1 where s is a new secret name being the seed for the private and public keys, and x_{cert} is A_0's certificate. The request message is defined to be just the pair of the signed identifier *rdp* and the certificate. We leave out the IP address of the destination from the request message because the destination in our simple model is fixed to be C_0. The reason why we leave out the nonce from a request is that they are solemnly there to help detecting recycling of messages which we do not care about in this paper.

The route discovery message is sent out on channel a_1. The process then waits for a triple (x_1, x_2, x_3) of inputs on channel a_2. Expectedly the inputs are in turn: i) a reply message signed by the destination and the forwarding process, ii) a certificate from the destination, and iii) a certificate from the forwarding process. First it is checked whether x_1 contains the identifier *rep*. Then the destination certificate x_2 is validated and afterwards the signature of the reply message is checked using the key from x_2. Next, the certificate x_3 is verified and the key in x_3 is used to validate the signed reply x_1. The successful reception of a reply is signaled by sending out *success* on channel a_3.

The names *rdp*, *rep*, and *success* are all free names of the model, and likewise are the channel names a_1, a_2, a_3 as well as the channel names b_1 and b_2 used in the remaining processes of the model.

The process B_0 defined in Figure 2 is supposed to receive a pair, x_1 and x_2, of a route discovery message sent on channel a_1 by A_0. It extracts the contents

$$
\begin{array}{l}
B_0 \overset{\text{def}}{=} \nu s.let\ x_{cert} = sign(pk(s), sk(t))\ in\ a_1(x_1, x_2).B_1 \\
B_1 \overset{\text{def}}{=} if\ rdp = get(x_1)\ then\ let\ x_{ikey} = get(x_2)\ \ in\ B_2 \\
B_2 \overset{\text{def}}{=} if\ check(x_{ikey}, x_2, pk(t)) = ok\ \ then\ B_3 \\
B_3 \overset{\text{def}}{=} if\ check(rdp, x_1, x_{ikey}) = ok\ \ then\ B_4 \\
B_4 \overset{\text{def}}{=} let\ x_{sreq} = sign(x_1, sk(s))\ in\ \overline{b_1}\langle x_{sreq}, x_2, x_{cert}\rangle.b_2(y_1, y_2).B_5 \\
B_5 \overset{\text{def}}{=} if\ rep = get(y_1)\ then\ let\ x_{dkey} = get(y_2)\ \ in\ B_6 \\
B_6 \overset{\text{def}}{=} if\ check(x_{dkey}, x_2, pk(t)) = ok\ \ then\ B_7 \\
B_7 \overset{\text{def}}{=} let\ x_{srep} = sign(x_1, sk(s))\ in\ \overline{a}_2\langle x_{srep}, x_2, x_{cert}\rangle
\end{array}
$$

Fig. 2. The intermidiary process B_0.

of x_1 and checks that it contains the identifier *rdp*, after which it verifies the certificate x_2 and validates that x_1 has been properly signed. Then it signs the request x_1 itself and sends out its forwarded request message on channel b_1. On channel b_2 it awaits a reply message, y_1 and y_2. It checks that the reply contains the identifier *rep*, validates the certificate y_2, and finally, it signs the reply message and outputs its forwarded reply message on a_2.

$$
\begin{array}{l}
C_0 \overset{\text{def}}{=} \nu s.let\ x_{cert} = sign(pk(s), sk(t))\ in\ b_1(x_1, x_2, x_3).C_1 \\
C_1 \overset{\text{def}}{=} let\ x_{sreq} = get(x_1)\ in\ if\ rdp = get(x_{sreq})\ then\ C_2 \\
C_2 \overset{\text{def}}{=} let\ x_{ikey} = get(x_2)\ in\ if\ check(x_{ikey}, x_2, pk(t)) = ok\ then\ C_3 \\
C_3 \overset{\text{def}}{=} if\ check(rdp, x_{sreq}, x_{ikey}) = ok\ then\ C_4 \\
C_4 \overset{\text{def}}{=} let\ x_{fkey} = get(x_3)\ in\ if\ check(x_{fkey}, x_3, pk(t)) = ok\ then\ C_5 \\
C_5 \overset{\text{def}}{=} if\ check(x_{sreq}, x_1, x_{fkey}) = ok\ then\ \overline{b}_2\langle sign(rep, sk(s)), x_{cert}\rangle
\end{array}
$$

Fig. 3. The destination process C_0.

The destination process C_0 defined in Figure 3 expectedly receives a forwarded request message, (x_1, x_2, x_3) from B_0 on channel b_1. It first tries to extract the identifier *rdp* from x_1 (using the destructor *get* twice), then it verifies the two certificates x_2 and x_3 and checks that the reply was properly signed, and finally it returns its reply message on channel b_2.

As for the request message also in the reply, the IP address of the receiver is left out because the receiver is fixed. Also, for the same reason as for the request messages since recycling is not an issue of analysis in this paper a nonce is not part of the reply message.

5 ProVerif

The tool *ProVerif* [4] is an automatic verifier for cryptographic protocols. It is based on logic programming. The tool is efficient in proving secrecy properties of protocols, in particular because it, as stated in [4], avoids the state space explosion problem thanks to an efficient solving algorithm.

In [1] the authors provide a translation from the variant of the Applied π-Calculus considered in this paper to ProVerif such that e.g. secrecy as defined in Section 3.3 can be verified automatically. In order to check whether a process P preserves the secrecy of some message M from a set of names S one only needs to specify P, the names S, the constructors and their arities, and the equations for the destructors. Then the attacker may use the following capabilities to learn more:

– For each constructor f of arity k, if the attacker knows $M_1, \ldots, M_k$ then it knows also $f(M_1, \ldots, M_k)$.
– For each destructor g and for each of its defining equations $g(M_1, \ldots, M_k) = M$, if the attacker knows $M_1, \ldots, M_k$ then it knows also M.
– If the attacker knows channel a and if $M_1, \ldots, M_k$ are sent on a, then the attacker knows also $M_1, \ldots, M_k$.
– If the attacker knows channel a and $M_1, \ldots, M_k$ then the attacker may send $M_1, \ldots, M_k$ on a.

```
fun sk/1.
fun pk/1.
fun sign/2.
reduc get(sign(m,sk(x))) = m.
free a.
private free s,t.
let P = let x = sign(t,sk(s)) in out(a, (pk(s),x)).
process P
```

Fig. 4. A sample ProVerif specification.

The process P defined by (3) in Section 3.2 is written using ProVerif syntax in Figure 4. We denote the three constructors *sk*, *pk*, and *sign* and their arities, we give the equation for the destructor *get*. We define that the channel name a is public and hence known by the adversary, but that the names s and t are private and thus not known by the adversary, and we specify the process P. Finally we tell that P is the process for investigation.

Given the specification in Figure 4 and asking now ProVerif the query:

```
query attacker:t.
```
(7)

meaning: “Will `P` preserve the secrecy of `t` from `a`” (or more intuitively: “Will the attacker know the name `t`”) we are told confirmatively that:

```
An attack has been found.
RESULT not attacker:t is false.
```

If we replace the name `t` by the name `s` in (7) we get as expected the affirmative answer:

```
RESULT not attacker:s is true.
```

6 Analysis

In this section we analyse the model of the ARAN protocol defined above in Section 4 by equation (6).

As a beginning we validate that a request message can be send from A_0 via node B_0 to C_0, and that the request is properly acknowledged by C_0 as a reply message returned in the opposite direction via B_0 to A_0. We may show this successful route request property indirectly by letting the attacker know only about the channel name a_3 and then check if it is told the secret *success*, i.e. we ask ProVerif the query:

$$Q = attacker : success. \tag{8}$$

ProVerif answers confirmatively: $!Q$ *is false*.

$$\boxed{\begin{array}{l}
{C_0}' \overset{\text{def}}{=} \nu s.let\ x_{cert} = sign(pk(s), sk(t))\ in\ c_1(x_1, x_2, x_3).{C_1}' \\
{C_1}' \overset{\text{def}}{=} let\ x_{sreq} = get(x_1)\ in\ if\ rdp = get(x_{sreq})\ then\ {C_2}' \\
{C_2}' \overset{\text{def}}{=} let\ x_{ikey} = get(x_2)\ in\ if\ check(x_{ikey}, x_2, pk(t)) = ok\ then\ {C_3}' \\
{C_3}' \overset{\text{def}}{=} if\ check(rdp, x_{sreq}, x_{ikey}) = ok\ then\ {C_4}' \\
{C_4}' \overset{\text{def}}{=} let\ x_{fkey} = get(x_3)\ in\ if\ check(x_{fkey}, x_3, pk(t)) = ok\ then\ {C_5}' \\
{C_5}' \overset{\text{def}}{=} if\ check(x_{sreq}, x_1, x_{fkey}) = ok\ then\ \overline{c}_2\langle sign(rep, sk(s)), x_{cert}\rangle
\end{array}}$$

Fig. 5. A revised destination process ${C_0}'$.

Next, suppose a situation in which the destination C_0 has moved out of the communication range with A_0 and B_0. This we may catch indirectly by enforcing that C_0 has no channel names in common with the other nodes, e.g. by letting the occurrences of the free names b_1 and b_2 in Figure 3 be replaced by new free names, say c_1 and c_2 respectively, as defined in Figure 5, and hence defining the revised ARAN model, *Aran*$'$, by:

$$Aran' \overset{\text{def}}{=} \nu t.(A_0 \mid B_0 \mid C_0') \quad . \tag{9}$$

If we want to check as to whether the attacker may now take over the role of the inaccessible destination ${C_0}'$ and generate false route request replies we may let the channel names b_1 and b_2 be public, thereby allowing the attacker to communicate with the node B_0. Letting also a_3 be public as before and asking again the query defined by (8) ProVerif now reports: $!Q$ *is true*, hence the attacker cannot generate false route request replies.

A slight modification to the seemingly safe changes to the ARAN model defined above by (9) turns out to pave the means for *spoofing attacks* where false routing information may occur. As before we expect that the destination ${C_0}'$ is outside the communication range of the other nodes in the network and further we now also assume that the attacker is within the ranges of both B_0 and ${C_0}'$. In the revised model (9) this comes into effect by letting not only a_3, b_1, and b_2 but also the new free names in Figure 5, c_1 and c_2, be public.

Asking then the query (8) ProVerif in this case answers: $!Q$ *is false*, i.e. it is indeed possible, although the destination ${C_0}'$ is out of range from the other nodes in the network, to obtain a successful path from A_0 to ${C_0}'$. From the diagnostic information provided by ProVerif it follows that the attacker has carried out a spoofing attack on the ARAN protocol, an attack where the adversary both impersonates B_0 and ${C_0}'$. It turns out that the attacker just relays messages sent to and from B_0 and ${C_0}'$, i.e. to be more precise: forwarded request messages supposed to be send from B_0 to ${C_0}'$ on channel b_1 are caught by the attacker and send unchanged to ${C_0}'$ on channel c_1, and likewise reply messages from ${C_0}'$ sent out on c_2 are forwarded unaltered by the attacker on b_2 to B_0. [7]

In the next section we provide a solution to the attack.

7 A solution proposal

The communication primitive in mobile ad hoc networks is broadcasting, hence any neighbor to a node can listen to all messages send by that node, at least if we assume that communication between nodes is bi-directional.

The *watchdog* principle was introduced in [12] as a means to detect routing misbehaviour between nodes. The watchdog idea is simply that when a node X sends a packet to a node Y it records the message sent. If X overhears Y forwarding its message then X knows that the messages has been successfully forwarded, if X does not within a certain time interval overhear the forwarding of its messages it registers that Y is acting erroneously.

We may adapt the watchdog idea similar to what has been done in [11] to help detect the spoofing attack outlined above. That is, we kind of reverse the checking in the watchdog such that a node X should instead check that it never overhears messages identical to the ones it has sent out. If that happens X knows that a neighbor is behaving in contradiction with the ARAN protocol.

To reduce the effect of malicious (or malfunctioning) nodes we suggest to extend the ARAN protocol such that any node X initiating or forwarding a request message r overhears all neighbors and records the ones correctly forwarding r

[7] In order to forward a message from a source to a destination without revealing its true identity the attacker must in real life impersonate the source by changing its MAC or IP address to that of the source.

within a certain time limit t. [8] Hence to each initiated or received request message a node maintains a *positive list* of neighbors. The entries in the list may be triples consisting of the IP address of the request initiator, the nonce of the request, and the IP address of the successful neighbor. The checking of correct forwarding may be carried out similarly to the current validation of forwarding in ARAN.

If a reply for request r is returned the receiving node should check that the reply is sent from a node in its positive list for r, and only if that is the case the node forwards the reply itself. We conjecture that this extension to the ARAN protocol will alleviate the relay attack outlined above.

However, although our suggested extension may work well for our demonstrated relay attack, it will not suffice if two neighbors X and Y are first outside each others transmission range when the reply message for a request r broadcasted by X is returned, because then X may have Y in its positive list for r and an attacker may therefore safely relay a reply from Y to X.

To overcome also this problem we suggest that a node should overhear the forwarding of any of its reply messages and record it in a *negative list* if the message is not forwarded according to the rules of ARAN, or if it does not receive a challenge (described below) within t time units. The entries in the list may be pairs of the IP address of the request initiator and the nonce of the request.

When a node X receives a forwarded reply

$$r' = [[rep, ip_A, n_A]_{K_D-}]_{K_Y-}, cert_D, cert_Y \tag{10}$$

it checks the message according to ARAN and if Y (i.e. (ip_A, n_A, ip_Y)) is in its positive list for the corresponding request then X sends back to Y the *challenge*

$$[ip_A, n_A]_{K_X-}, cert_X, \{n\}_{K_Y+} \tag{11}$$

where n is a new nonce and $\{n\}_{K_Y+}$ is the encryption of n by the public key of Y. X then awaits to have n sent back encrypted by its own public key. [9] Y only returns n in case r' (i.e. (ip_A, n_A)) is not in its negative list and if it receives (11) within t time units after submitting (10). If X receives $\{n\}_{K_X+}$ then it proceeds according to ARAN.

With these extensions to ARAN, if X receives (10) directly from Y, and if Y is in X's positive list for the corresponding request, then the reply is forwarded according to ARAN along the back path from X. Clearly, when X in this case makes a challenge to Y it must happens fast enough to satisfy the time limit t.

Suppose instead that an attacker wants to exploit that X is not within Y's transmission range when the reply (10) is transmitted between the two. Since

[8] Suppose X and Y cannot hear each other. The time limit t should be devised such that it is not possible within t time units for a node A to: receive a message m from X; move to let A and Y, but not A and X, be within transmission range; forward m to Y and pick up Y's forwarding of m, say m'; and finally move back to let X, but not Y, be within A's transmission range where then m' is revealed to X.

[9] The public keys of X and Y may be retrieved from their certificates.

the attacker within t time units cannot forward r' to X without Y listening and in return obtain a proper challenge to display to Y then Y registers r' in its negative list. Hence X never receives its nonce and never forwards r' so the attack fails.

8 Conclusion

In this paper we have applied the automatic verification tool ProVerif on an Applied π-Calculus model of the ARAN protocol for secure routing. To the best of our knowledge we are the first to carry out formal automatic verification of that kind of protocols.

We conjecture that the ARAN protocol, despite its use of certificates and signing of messages, is erroneous in that an adversary may inject false routing information by a man-in-the-middle spoofing attack. We propose an addendum to the protocol that we claim to be a guard against the identified attack.

The attack was identified by a manual modification of the ARAN model to impose the necessary node mobility. Ideally an automatic verification tool should by itself identify the node movements that facilitates attacks, but that seems to be a promising research direction. To help identifying this kind of attacks a modeling language therefore should support explicit node mobility, and broadcasting of messages should be a communication primitive. It seems also to be advantageous to have an explicit notion of location and a supporting metric in such a language since broadcasted messages have a limited transmission range. Moreover, an explicit account of time in the language is needed in order to support our needs for dealing with time constraints.

Perhaps equally important is that a revision of the standard Dolev-Yao attacker model [6] is needed, it seems unfeasible that an adversary can overhear all communication in ad hoc networks, but on the other hand the overheard channels should be able to change over time.

As future work it would be of interest to verify our suggested solution.

References

1. Martín Abadi and Bruno Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. *Journal of the ACM*, 52(1):102–146, January 2005.
2. Martin Abadi and Cedric Fournet. Mobile vales, new names, and secure communication. In Hanne Riis Nielson, editor, *28th ACM Symposium on Principles of Programming Languages*, pages 104–115, London, UK, January 2001. ACM.
3. Karthikeyan Bhargavan, Davor Obradovic, and Carl A. Gunter. Formal verification of standards for distance vector routing protocols. *J. ACM*, 49(4):538–576, 2002.
4. Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In Steve Schneider, editor, *14th IEEE Computer Security Foundations Workshop*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society Press.

5. R. de Renesse and A.H. Aghvami. Formal verification of ad-hoc routing protocols using spin model checker. In *IEEE MELECON*, pages 331–340, Dubrovnik, Croatia, May 2004. IEEE Computer Society.
6. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information and Theory*, 29(2):198–208, 1983.
7. Y. Hu, A. Perrig, and D. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks, 2002.
8. Yin-Chun Hu, David Johnson, and Adrian Perrig. SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks. In *Proceedings of MobiCom*, September 2002.
9. David B. Johnson, David A. Maltz, and Josh Broch. DSR: The dynamic source routing protocol for multihop wireless ad hoc networks. In C.E. Perkins, editor, *Ad Hoc Networking*, chapter 5, pages 139–172. Addison-Wesley, 2001.
10. G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. *Software - Concepts and Tools*, 17:93–102, 1996.
11. John Marshall, Vikram Thakur, and Alec Yasinsac. Identifying flaws in the secure routing protocol. In *Proceedings of The 22nd International Performance, Computing, and Communications Conference (IPCCC 2003)*, pages 167–174. IEEE, April 2003.
12. Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 255–265, New York, NY, USA, 2000. ACM Press.
13. Robin Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, May 1999.
14. P. Papadimitratos and Z.J. Haas. Secure routing for mobile ad hoc networks. In *Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, January 2002.
15. Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.
16. Charles E. Perkins and Elizabeth M. Royer. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.
17. Amir Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint relaying: An efficient technique for flooding in mobile wireless networks. Technical Report Research Report RR-3898, INRIA, February 2000.
18. Kimaya Sanzgiri, Daniel LaFlamme, Bridget Dahill, Brian Neil Levine, Clay Shields, and Elizabeth M. Belding-Royer. Authenticated routing for ad hoc networks. *IEEE Journal on Selected Areas in Communication, special issue on Wireless Ad hoc Networks*, 23(3):598–610, March 2005.
19. Oskar Wibling, Joachim Parrow, and Arnold Pears. Automated verification of ad hoc routing protocols. In *FORTE*, pages 343–358, Madrid, 2004. IFIP Internation Federation for Information Processing.
20. Manel Guerrero Zapata. Secure ad hoc on-demand distance vector routing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(3):106–107, 2002.