THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS & SCIENCES

AN ANALYSIS OF THE SECURE ROUTING PROTOCOL FOR MOBILE AD HOC NETWORK ROUTE DISCOVERY: USING INTUITIVE REASONING AND FORMAL VERIFICATION TO IDENTIFY FLAWS

By

JOHN D. MARSHALL, II

A Thesis submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Spring Semester, 2003

The members of the Committee approve the thesis of John D. Marshall, II defended on April 23, 2003.

Alec Yasinsac
Professor Directing Thesis

Mike Burmester
Committee Member

Xin Yuan
Committee Member

The Office of Graduate Studies has verified and approved the above named committee members.

To my parents. Thank you for bestowing me an equal blend of your talents and the guidance to use those talents wisely.

ACKNOWLEDGMENTS

While this section is perhaps least significant to readers, this author would be remiss not to extend his sincerest gratitude to those without whose help this work would not have developed. I extend a big thank-you to the members of our security research group who endured what probably seemed like endless discussions of SRP and CPAL-ES. You all twisted my mind with your insights and helped me to develop my ideas and grow along the way.

A special thanks to Stephen Carter for introducing SRP and likewise to Vik Thakur for his thoughts on further attacks. I am indebted to Alec Yasinsac for his suggestions and revisions to this document. More importantly, I thank him for his consistent guidance and encouragement throughout my graduate experience. I am proud to call Alec and his family dear friends.

Finally, I would like to thank those who helped me maintain sanity throughout this process. Among this group are existing friends and those with whom I crossed paths along the way. These friends and acquaintances need no specific mention to know they are appreciated. Without them this work may have been possible to complete, but it would not have been near as much fun.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With this work we analyze the *Secure Routing Protocol (SRP)* proposed by Papadimitratos and Haas as a means for securing source-driven route discovery in ad hoc networks. We discuss flaws in SRP's design and expose SRP's shortcomings with respect to its security goals. We analyze SRP using both the canonical BAN logic analysis for cryptographic protocols and the CPAL-ES formal methods suite.

Formal methods work in broadcast paradigms is a burgeoning research area. As such, we discuss the challenges this environment poses for formal methods analysis and suggest changes to CPAL-ES that will assist future ad hoc network protocol analyses.

We introduce an attack which demonstrates SRP's vulnerabilities and our *bloodhound* solution. We abstract the attack and conclude that protocols developed in the likeness of SRP are equally vulnerable. Finally, we point out SRP's unrealistic assumptions that make it unfit for use in practical MANET applications.

CHAPTER 1

INTRODUCTION

Mobile ad hoc networks (MANETs) are gaining popularity as an attractive, and in some situations a necessary, alternative to traditional wired networks. MANETs provide a means of peer-to-peer communication without a pre-existing infrastructure. Their uses range in scale from servicing rural mountainous regions to provisions for communications in emergency or battlefield situations [1, 2].

Because of their natural characteristics, MANETS utilize message relay to deliver messages. For efficiency, many have proposed routing algorithms to uniquely support this purpose [3, 4]. Recently, a flurry of research has been directed at protecting routing in this environment [5, 6, 7]. In this work, we illustrate the complexities involved with evaluating MANET security protocols by analyzing the proposed *Secure Routing Protocol* (*SRP*) [8].

1.1 Cryptography and Security

Because MANETs are put into practice, it behooves us to find ways to protect these networks with the common security goals of confidentiality, authentication, and integrity. It is widely known that cryptography is used to provide privacy of communicated information; indeed, cryptography is often loosely defined as the art of secret writing. For our purposes, however, cryptography has a much more developed definition. Table 1.1 lists the three security features cryptography provides with a brief description. We address each of these in more detail below.

Confidentiality fits well with the colloquial use of the term cryptography. That is, confidentiality is concerned with maintaining the secrecy of message contents. Being the most popular goal of cryptography, it is also the most widely studied. There are numerous mechanisms which provide privacy, referred to as cryptosystems. Cryptosystems run the gamut with respect to strength (i.e., their difficulty to break). For strong cryptosystems,

Table 1.1. Security goals of cryptography.

Goal	Description
Confidentiality	provide privacy and secrecy
Authentication	identify origins of message
Integrity	detect message modification

it is generally accepted that the choice of key and not the obfuscation of the algorithm is what makes the resulting cipher-text hard to decipher. All cryptosystems have well-defined encryption and decryption algorithms that have these properties.

$$\begin{aligned} c &= e[msg]k, \\ msg &= d[c]k', \text{ or alternatively} \\ msg &= d[e[msg]k]k'. \end{aligned}$$

With encryption key k and matching decryption key k', decryption function d under key k' performs the reverse operation of encryption function e under key k.

With authenticity, the goal is to determine the origin of a message. Most often this is accomplished by the originator disclosing some knowledge that only it possesses. Passwords and biometrics serve this purpose, the former considering what an entity knows, the latter what an entity physically possesses. An alternative approach is for both communicating principals to establish a trust relationship with a trusted third party. The principals then authenticate utilizing this shared trust in the third party.

Integrity is concerned with whether message contents have been modified. Methods for protecting message integrity devise strategies to detect when a message has been tampered with. A common technique is to attach a digest (or hash) of the message along with the message itself. As the name suggests, the digest is a condensed fixed-length output of the input message. Message digests have many special properties [9], most important of which is one-wayness. This property states that given a hash function $h : \mathcal{X} \rightarrow \mathcal{Y}$ and an element $x \in \mathcal{X}$, it is computationally infeasible to find $x' \in \mathcal{X}$ with $x \neq x'$ such that $h(x) = h(x')$. For our purposes, this expresses that it is sufficiently difficult to alter a message so that this new message produces the same digest as the unaltered message. A receiver verifies message integrity by executing the hash function on the received message and comparing the result to the received hash.

The goals of authenticity and integrity are often combined using a keyed-hash. The resulting computation is called a *Message Authentication Code* (*MAC*). A MAC is computed as follows.

$$\text{MAC} \;=\; h(msg, k)$$

While the hash function is publicly known, only two parties who share key k can produce the MAC. The properties of hash function h protect integrity while key k provides authentication.

1.2 Security Protocols

As we intend to study security protocols, it serves us well to formally define the terms that we rely upon. Security expert Bruce Schneier defines protocols as three-pronged structures [10]:

1. a series of steps

2. 2 or more participants (principals)

3. well-defined goals

In many ways protocols resemble algorithms, in that they follow a process and accomplish some task. This does not comprise a protocol, however, without the participation of two or more principals.

Cryptographic (more recently, security) protocols employ the cryptographic techniques described in Section 1.1 to accomplish the security goals of confidentiality, authenticity, and integrity. The study of security protocols demonstrates that the process of achieving these security goals is just as important as the mechanism itself; stated concisely, the *how* is equally as important as the *what.* One could choose the strongest cryptosystem, but if a well-designed security protocol is not followed, security can easily be compromised.

Cryptographic techniques are used in protocols to prevent malicious principals from thwarting the goals of the protocol. While this purpose seems clear, the means to this end are not always so crystalline. As we will see with SRP, attacks and vulnerabilities are often subtle and all too frequently the result of insufficient analysis on the part of the designers.

1.3 Secure MANET Routing

The goal of SRP is to successfully establish a route between source and destination nodes in an ad hoc network of untrusted nodes. This is achieved through source-driven route discovery. SRP is a security extension for existing insecure MANET routing protocols. The novelty of SRP is that only the two end nodes (source and destination) must have a security association, which includes a key. Where many MANET protocols assume the presence of a public-key infrastructure (PKI) or group-key establishment, SRP makes no such assumptions about intermediate nodes. Still, the designers claim that after SRP executes the source and destination are **guaranteed** that the *route* between them is non-corrupted. With this as its goal, our analysis shows that SRP is flawed.

We begin in Chapter 2 with an introduction to the Secure Routing Protocol. In Chapters 3 and 4 we analyze SRP using the canonical BAN logic as well as CPAL-ES formal methods. We discuss an overlooked attack on SRP and its solution in Chapters 5 and 6. In Chapter 7 we argue that any security protocol designed in the likeness of SRP is inherently flawed. We conclude this work by suggesting future research directions and a summary of our analysis.

CHAPTER 2

THE SECURE ROUTING PROTOCOL

Route discovery in MANETs has been a key area of focus in recent years, with the Ad-hoc On-demand Distance Vector routing (AODV) [11] and Distance Vector Routing (DSR) [1] algorithms emerging as *de facto* protocols. Rather than their standard pro-active counterpart in wired networks, AODV and DSR are reactive protocols that establish routes on an "as-needed" basis. Route discovery is source-driven with a query and response scheme. Both protocols share the benefits of simplicity, efficiency, and low resource consumption.

The two protocols differ with respect to their storage of routing information. In AODV, nodes store only the next node along the route to a destination. DSR, on the other hand, stores the entire route at each node. The Secure Routing Protocol (SRP) is designed in the likeness of DSR.

SRP, in fact, naturally extends DSR and similar protocols such as the Zone Routing Protocol (ZRP) [12] and the Interzone Routing Protocol (IERP) [13]. While these latter protocols pay little attention to the security issues discussed in Chapter 1, SRP's goal is to establish routes which correctly identify the network topology even in the face of malicious intruders. The three operating assumptions of SRP are:

1. communication between nodes is *bi-directional*, defined in [2] to imply symmetric node communication. That is, if A can receive from B at time t then the converse also holds (namely, B can receive from A at time t);

2. a *Security Association (SA)* exists between the source S and target T. The SA provides authentication by the pre-established shared key $K_{S,T}$ between S and T; and

3. the protocol runs in an environment with non-colluding nodes.

SRP's insistence that only the two end nodes require a shared key makes the approach novel. However, as we will show, the fact that there is little control over the activity of intermediate, routing nodes is cause for concern.

We provide a working description of the steps involved in SRP, while remaining faithful to the authors' design [8]. SRP has four fundamental components: (1) route request, (2) query propagation, (3) route reply, and (4) reply validation.

2.1 Route request

For source node S to initiate a route discovery to target node T, it issues a route request in a query packet. The following four fields are contained within the SRP header:

1. $Q_{seq} \rightarrow$ a monotonically increasing *Query sequence number* that S maintains for each T between which it attempts to establish a route. This field is used by T to recognize outdated or replayed route requests.

2. $Q_{ID} \rightarrow$ a 32-bit random number that provides distinguishability of requests.

3. *Message Authentication Code (MAC)* $\rightarrow$ a 96-bit keyed-hash of the IP header, Q_{seq}, and Q_{ID}. The MAC is generated with the key $K_{S,T}$ shared by S and T.

4. *route* $\rightarrow$ initialized with the source address S.

The MAC excludes those fields within the IP and SRP headers that may change during packet propagation. The authors claim this use of a MAC renders "the scheme efficient and scalable." However, we will demonstrate that it makes SRP vulnerable to attack.

With the route request packet initialized by the source, it is transmitted by broadcast.

2.2 Query propagation

Nodes within the transmission range of source node S and are not target node T are called intermediate nodes. These nodes act as relays for the route request, allowing it to traverse the ad hoc network.

Upon receipt of a route request, intermediate nodes extract the source and destination addresses from the IP header and the Q_{ID} field from the SRP header; it uses the source and destination addresses to maintain an entry in its query table with a Q_{ID} attribute. If a source-target entry with a matching Q_{ID} exists, the packet is discarded as a replayed or previously processed route request. Otherwise, the intermediate node updates the route field by appending its IP address. It then forwards the route request by broadcast.

This query propagation is repeated by subsequent intermediate nodes until the packet reaches its target.

2.3 Route reply

When the route request reaches target node T it is verified. T first checks the source address in the IP header to determine if it shares an SA with the query initiator. If it does, T compares Q_{seq} from the SRP header to S_{max}, a maximum sequence number T maintains for each of its SA partners. If $Q_{seq} \leq S_{max}$, the query packet is discarded as a replayed or previously processed request. Otherwise, T computes the MAC with Q_{seq}, Q_{ID}, the IP header, and $K_{S,T}$. T then compares the received MAC with the one it computes to verify message integrity with respect to the included fields.

At this point, T composes the route response. The response packet includes the four fields of Section 2.1, with one modification to the MAC field. T will include the accumulated route field within the response's MAC. The response is then unicast along the reverse of the identified route.

Intermediate nodes will perform the checks of Section 2.2 and propagate the response accordingly.

2.4 Reply validation

When S receives a response packet, it verifies the source, destination, Q_{seq}, and Q_{ID} fields to determine the packet's legitimacy in response to a valid query. S then compares the route accumulated during the response packet's propagation to the reverse of the *route* field identified by T. If the two routes do *not* match, the response is dropped due to detected malicious activity. Otherwise, S computes the MAC from Q_{seq}, Q_{ID}, the IP header, and $K_{S,T}$. It then compares the MAC it computes with the MAC received in the response packet.

If the MACs match, S accepts the route response packet as non-corrupted and deems the route a legitimate path to send traffic between itself and T.

2.5 Remarks on SRP

We argue that using only a MAC and an SA between two end nodes does not ensure that the identified route between S and T actually exists. The vulnerability exists for two related reasons. First, the protocol does not effectively regulate the activity of intermediate nodes. Second, as a result of unregulated intermediate node activity, target node T has no way of verifying any data altered or accumulated by intermediate nodes.

With regards to the first, we note that the designers intended matching forward and reverse routes to regulate intermediate node behavior. The designers argue that this, coupled with the first-in wins strategy whereby the route request first received by T supersedes all subsequent requests, sufficiently ensures security. Still, intermediate nodes are not forced to append, for example, their IP addresses. Our attack exploits this fact. With regards to the second, because T cannot trust data altered by intermediate nodes it has no way to verify that the *route* field is, in fact, legitimate. We elaborate on these issues more when we introduce our attack in Chapter 5.

CHAPTER 3

BAN LOGIC ANALYSIS OF SRP

BAN logic [14] is a logic of belief used to infer properties about security protocols. BAN comprises its own notation of predicates and rules of inference. In Table 3.1 we provide a summary of some common predicates. BAN predicates are used to express beliefs and intentions of participants in a given protocol. There are three rules of inference that allow analysts to deduce properties of a given protocol from the predicates. For our analysis, we are concerned with the inference rules of jurisdiction and message meaning given below; the third may be found in [14]. For **jurisdiction** we have

Table 3.1. BAN logic notation.

Notation	Description
$P \lhd X$	P is told X
$P \ni X$	P possesses X
$P \mid\sim X$	P once said X
$P \mid\equiv X$	P believes X
$P \mid\equiv \sharp X$	P believes X is fresh
$P \overset{K_{P,Q}}{\longleftrightarrow} Q$	$K_{P,Q}$ is a goodkey between P and Q
$P \mid\Rightarrow X$	P has jurisdiction over X

$$\frac{((A\mid\equiv S\mid\equiv X),(A\mid\equiv S\mid\Rightarrow X))}{(A\mid\equiv X)}. \tag{3.1}$$

In general, the numerator expresses the predicate(s) which must be true to infer the predicate(s) in the denominator. In this case, if A believes that S believes X **and** A believes that S controls (has jurisdiction over) X, then we can infer that A believes X. For **message meaning** we have

$$\frac{((A \overset{kab}{\longleftrightarrow} B),(A \lhd \{X\}kab)}{(A\mid\equiv B\mid\sim X)}. \tag{3.2}$$

That is, if A and B share a goodkey kab and A is told $\{X\}kab$, then we can infer that A believes that B once said X.

A common simplification axiom that allows inference of fields within concatenated values is

$$\frac{P|\equiv (X,Y)}{P|\equiv X}.$$

Succinctly, this states that if we believe a concatenated message we also believe its constituent parts.

The process of coding of protocol in BAN logic notation has three steps: idealization, annotation, and deduction. In idealization, the analyst translates actions on data into beliefs and intentions of the involved principals. This requires a macroscopic view of the purpose of the protocol. As an example, the following message

$$A \rightarrow B : \{B, kab\}kbs$$

could be idealized to

$$A \rightarrow B : A \overset{kab}{\longleftrightarrow}_B \ . \tag{3.3}$$

Annotation uses simplification rules or intuitive reasoning to find further predicates that may be helpful when combined with the inference rules. To continue our example, Equation 3.3 may be annotated by

$$B \lhd \{A \overset{kab}{\longleftrightarrow} B\}kbs.$$

With these predicates one can now use the three inference rules to deduce properties about the protocol.

3.1 The designers' BAN analysis

In their introduction of SRP [8] the designers present a BAN logic analysis of the protocol. They decompose SRP into its two fundamental messages, the route request (query) and the route reply (response). These two messages can be represented by

$$\begin{array}{ll} S \rightarrow T & Q_{S,T}, H(Q_{S,T}, K_{S,T}) \\ T \rightarrow S & R_{S,T}, route, H(R_{S,T}, route, K_{S,T}). \end{array}$$

$Q_{S,T}$ is the route request header information which includes the source, target and Q_{seq}; $R_{S,T}$ is the route reply containing the same three fields. Recall, Q_{seq} links $R_{S,T}$ to $Q_{S,T}$. In these messages, the MAC value is represented by H(<fields>, <key>).

The initial assumptions of SRP are given below in BAN logic notation.

$$S \ni K_{S,T},\ S|{\equiv} S \xleftrightarrow{K_{S,T}} T$$
$$T \ni K_{S,T},\ T|{\equiv} S \xleftrightarrow{K_{S,T}} T$$
$$S \ni Q_{seq},\ S|{\equiv}\ \sharp Q_{seq}$$
$$T \ni N^p_{S,T}.$$

We assume the reader has little familiarity with BAN, so we also provide the following verbose description of the assumptions. Both S and T possess the secret key $K_{S,T}$ and believe that it is a good key. Furthermore, S possesses sequence number Q_{seq} and believes it is fresh. Lastly, T possesses a list of all sequence numbers $N^p_{S,T}$ that S has sent with previous route requests.

When T receives the forward route request originating at S,

$$\frac{T \lhd (Q_{S,T}, H(Q_{S,T}, K_{S,T}))}{T \ni (Q_{S,T}, H(Q_{S,T}, K_{S,T}))},$$

from which one can simplify

$$\frac{T \ni (Q_{S,T}, H(Q_{S,T}, K_{S,T}))}{T \ni Q_{seq}}.$$

Recall that $Q_{S,T}$ contains Q_{seq} within it. T uses Q_{seq} to determine the freshness of the route request. If $Q_{seq} \notin N^p_{S,T}$ then T believes the message is fresh. T will then compute the MAC from the fields within $Q_{S,T}$ and check its validity with $H(Q_{S,T}, K_{S,T})$ it receives. If the message is fresh and the MACs match, then T believes that S once said (transmitted) both the sequence number and the MAC. In BAN logic notation,

$$T|{\equiv}\ S|{\sim}\ (Q_{S,T}),\ T|{\equiv}\ S|{\sim}\ (H(Q_{S,T}, K_{S,T})).$$

With these beliefs T prepares a response packet and includes the *route* field accumulated during query propagation by intermediate nodes.

Upon S's receipt of the forwarded response packet,

$$\frac{S \lhd (R_{S,T}, route, H(R_{S,T}, route, K_{S,T}))}{S \ni (R_{S,T}, route, H(R_{S,T}, route, K_{S,T}))}$$

and subsequently

$$\frac{S \ni (R_{S,T}, route, H(R_{S,T}, route, K_{S,T}))}{S \ni Q_{seq}}.$$

Again, recall that $R_{S,T}$ contains Q_{seq} within it. Because S believes the freshness of Q_{seq}, S also believes the freshness of $(Q_{seq}, route)$ and $R_{S,T}$ which contains both fields. With the predicates

$$S \lhd H(R_{S,T}, K_{S,T}), S \overset{K_{S,T}}{\longleftrightarrow} T,$$

one can use the message meaning rule from Equation 3.2 to infer

$$S| \equiv T| \sim R_{S,T}.$$

At this point the designers claim "S believes that the entire route reply datagram originates from T and is fresh and, **trivially**, that T has constructed *route*" [8] (emphasis on "trivially" ours). While we agree with the former statement, we first showed in [15] the latter claim to be false. That is, T does not in fact construct the *route* field as the designers suggest; rather, T accepts the accumulated route based on misplaced trust.

It is evident that T once said (transmitted) the *route* field. In no way does this guarantee the accuracy of what T said let alone the security of the protocol as a whole. We expand our argument in the next section.

3.2 Critique of BAN analysis for SRP

In their seminal paper introducing BAN logic for analysis of authencation protocols, Burrows et al. note that BAN was not designed to consider the impact of clear-text messages in protocols. Specifically, they state:

> The idealized protocols of the examples given ... do not include clear-text message parts ... We have omitted clear-text communication simply because it can be forged, and so its contribution to an authentication protocol is mostly one of providing hints as to what might be placed in encrypted messages [14].

So while clear-text messages can be used by analysts to better understand the intent of a protocol, the authors are quite clear that clear-text messages can compromise security.

That the *route* field in the SRP header is sent in the clear is cause for concern. While use of the *route* field may be legitimate when linked to an encrypted message part, because the route is created in transit no such link exists between it and the MAC. Furthermore, no single node can claim responsibility for the route's correctness. As a result, its stand-alone use is errant.

However, S still uses the reverse of the *route* to identify the contents of the MAC. This despite T having placed the route within the MAC without any means to verify its integrity. The designers' word choice that T has "constructed" the route is misleading; T has simply forwarded the accumulated route from a query request it verifies as non-corrupted. Our attack in Chapter 5 demonstrates how T can erroneously reach such a conclusion.

The designers' inferences from Section 3.1 are necessary but insufficient to logically reach their conclusion. To say that T has constructed the route, some node (perhaps T itself) must necessarily have jurisdiction (control) over the *route* field so that S can believe the route is legitimate. According to the jurisdiction rule of Equation 3.1, we require

$$\frac{S|\equiv T|\Rightarrow \mathit{route} \text{ and } S|\equiv T|\equiv \mathit{route}}{S|\equiv \mathit{route}}.$$

That is, S must believe both that T has control over the route and also that T believes the route for it to truly **believe** that *route* is legitimate. While this may stretch the limits of BAN logic (see Section 3.3), the designers do not attempt to claim that T believes it **controls** *route*, let alone that S believes that either T believes or controls the route. As a result, S has no grounds on which to base its belief that the *route* field is legitimate.

3.3 Limitations of BAN logic

We not only wish to illustrate that the designers' conclusion about the source node's belief in the route's validity is false, we also argue that any such belief does not in itself constitute a claim that the protocol is secure. This work was first introduced in [16].

It is widely accepted that logics of belief provide no guarantee of security [17, 18, 19]. Security protocol analyst Paul Syverson states, "The goal of a logic such as that of Burrows, Abadi and Needham is to evaluate the trust that may rightly be placed in a protocol by legitimate participants" [18]. It does not, however, say anything about the security of such protocols in hostile environments. As the designers of BAN themselves note, "The restrictive, operational notion of belief that we have adopted would certainly be harmful in the study of security protocols" [14]. Furthermore, BAN logic is an incomplete system. As such, a BAN logic proof showing success or failure does not mean that the protocol is insecure; rather, it points to a potential problem. So, we view the inability of BAN to find SRP's attack not as a *flaw* of BAN logic, but rather a *misuse* of BAN logic. As Syverson succinctly states, "BAN deals only with trust and not security" [18].

Assessing protocol security represents a difference in perspective from that of trusted principal beliefs. As an analyst, one must view the protocol from the eyes of a malicious adversary. In this way one goes beyond the beliefs of legitimate participants to identify potential vulnerabilities that intruders can exploit. With untrusted intermediate nodes, surely we must consider the actions of illegitimate nodes in the network in which SRP executes. BAN cannot reason about such illegitimate principals.

Finally, we note that BAN logic was intended for use with protocols in traditional, wired networking environments. In such networks, true point-to-point communications is (or can be) achieved. However, the very nature of ad hoc networks is broadcast and dependent upon routing amongst potentially malicious nodes. BAN has no mechanisms for handling the "leap-frogging" technique of SRP in accumulating the route. We feel this, too, stretches BAN beyond its intended purposes.

CHAPTER 4

CPAL-ES ANALYSIS OF SRP

As we discussed in Chapter 1, appreciation for the value of protocol analysis stemmed from the realization that even simple protocols can be attacked in very subtle and clever ways [20]. As a result, a number of techniques have been employed to analyze security protocols. We have already seen BAN logic analysis in the previous chapter. In this chapter, we discuss a technique that employs formal methods.

With formal methods, the analyst has a rigorous proof mechanism which can be used to determine if a protocol attains its security goals. Borrowing concepts from programming language design, the analyst develops a formal syntax to represent the protocol steps and a semantics to reason about the meaning of these steps. For our study, we have chosen the Cryptographic Protocol Analysis Language Evaluation System (CPAL-ES) [21], an existing formal methods environment developed specifically for security protocol analysis. In the following sections, we introduce CPAL-ES and our use of it in the analysis of SRP.

4.1 Syntax of CPAL-ES

CPAL-ES uses protocol specifications in standard notation as a guideline for creating a formal syntax. The fundamental aspect of any protocol is its send operator, but CPAL-ES makes the matching receive operator equally important; where standard notation assumes a receive succeeding a send, CPAL-ES makes such assumptions explicit.

CPAL-ES uses a queue structure to maintain the ordering of sends and receives that works as follows. A sender uses a send statement that places the sent message on the receiver's queue. The send should be viewed as taking a value in the sender's address space and making it *available* to the receiver. The receiver explicitly extracts this value from the front of the queue with a receive statement; this places the value in the receiver's address space. CPAL-ES's queue structure enforces an ordering of alternating send and receive statements.

We use the following example to further illustrate the send and receive statements, as well as provide an introduction to the syntax of CPAL-ES.

```
A: => B(msg);
B: <- (msg);
```

CPAL-ES defines send statements in one of two ways: secure- and insecure-sends. The former is most commonly used. The secure-send assumes the establishment of a secure channel between the sender and receiver. In the example above, `A` securely sends (`=>`) the message `msg` to `B`. The subsequent receive (`<-`) statement is issued by `B`. Notice that a CPAL-ES statement begins with a principal, then a separating colon, the statement the principal will execute, and finally an ending semi-colon.

The insecure-send (`->`) is used for those instances when a secure channel is either not available or cannot be assumed. The purpose of the insecure-send is to allow for malicious intermediate node activity. Regardless of the intended recipient, in CPAL-ES an insecure-send must always be followed by a receive statement by malicious intruder `I`. Consider the following example.

```
A: -> B(msg);
I: <- (msg);
I: => B(dummy_msg);
B: <- (msg);
```

The code is similar to our first example, only that `A`'s secure-send is replaced with an insecure-send. Intruder `I` can manipulate the received message in any way, or simply replace it as is the case here. `I` may then forward a message onto the intended recipient `B`.

A common operation in many protocols is concatenation. CPAL-ES supports this action with the following syntax.

```
A: msg := <part1,part2>;
```

Here we see that `A` assigns the concatenation of `part1` and `part2` to `msg`. Concatenation of multiple fields is permissible, with each field separated by a comma. The reverse operation (separation of concatenated fields) is expressed by the following.

```
A: (part1,part2) := msg;
```

Functions in CPAL-ES are globally defined. The design strategy here was to permit two principals invoking a function with identical parameters to be returned the same value. This adds simplicity to the implementation, but also supports mechanisms used in security protocols. For example, hash functions (such as the one below) are typically publicly defined. As we discussed in Section 1.1, hash function's security is achieved through a parameterized secret key.

```
A: hash(parm1,parm2);
```

Naturally, CPAL-ES supports symmetric and asymmetric cryptography. However, the four features described above (send, receive, concatenation and functions) are the only ones we employ in our analysis. For a more thorough description of the CPAL-ES syntax, see [21, 22].

4.2 Semantics of CPAL-ES

CPAL-ES uses weakest precondition logic to reason about security protocols, building on Hoare logic [23, 21]. We illustrate this concept with an example. Suppose we execute the statement ($y := x + 3$) and that we want the postcondition ($y == 7$) to be true after statement execution. The goal of Hoare logic is to determine what precondition must be true *before* statement execution for the postcondition to be true *after* statement execution. The problem takes the general format

$$P\{S\}Q$$

where S is ($y := x+3$), Q is ($y == 7$), and P is the weakest precondition to be determined. To solve for P, we work in reverse and perform substitutions as follows. For a given postcondition Q, replace variables from Q with equivalent right-hand values from statement S. In our example, Q derives

$$(y == 7) \quad \Longrightarrow \quad (x + 3 == 7).$$

To find the weakest precondition P, simply solve for x. In this case, P is ($x == 4$).

As one can see, Hoare logic requires a bottom-up approach. It is extended for multiple statements with the following concatenation rule,

$$P\{S_1, S_2\}Q \quad \Longrightarrow \quad P\{S_1\}R,\, R\{S_2\}Q$$

where R represents some intermediate condition. In simple terms, R serves as both the postcondition of statement S_1 *and* the precondition of statement S_2.

Postconditions in CPAL-ES are expressed with assertion statements. Assertions represent the goals of a security protocol. Assert statements take two forms: `assert()` and `gassert()`. The former is used for variables within the asserting principal's address space, while the latter allows for variable comparisons across address space boundaries. For example, to assert the equality of the variable `val` in `A` and `B`'s address spaces, the following statement could be used.

```
global: gassert(A.val == B.val);
```

Notice that variables with the global assert must be prepended with the principal in whose address space they belong; we call such variables fully-qualified.

Analysis performed by CPAL-ES steps backward through a protocol to determine the weakest preconditions that must be true for the postconditions to logically be deduced – that is, for the security goals to be achieved. Once the weakest preconditions are determined, they can be expressed as assumptions with the assume statement. For example, it is common to see a key shared by `S` and `T` be assumed equal with the following.

```
global: assume(S.k == T.k);
```

Assume statements also use fully-qualified variables to distinguish principal address spaces. Once a statement is assumed, further statements can be logically derived from it; this often greatly simplifies CPAL-ES analysis.

In this way, CPAL-ES does not directly identify protocol flaws or vulnerabilities. Rather, CPAL-ES forces the analyst to explicitly state (with assume statements) the necessary conditions for a protocol to attain it security goal(s). Such assumptions are either tacit or have yet to be considered. As we have alluded to already, tacit assumptions are dangerous in security protocols; they leave room for interpretation and the possibility for injected error.

Authors' assumptions should be clearly expressed to ensure proper protocol implementation and use. Assumptions that have yet to be considered provide insight into the design and strength of a protocol, and they may even identify vulnerabilities. Our analysis in Section 4.4 demonstrates such a vulnerability with SRP.

4.3 CPAL-ES Encoding of SRP

CPAL-ES was developed with wired security protocols in mind. While many of the CPAL-ES design features carry over to the wireless environment, we encountered many difficulties in our encoding of SRP. We address these issues in this section, and in Chapter 8 we suggest adaptations to CPAL-ES to assist wireless protocol analysis.

Little work has been done with formal methods in wireless environments. The ad hoc nature of this environment introduces complexities. In true MANET route discovery protocols, nodes can assume very little about network topology. In most cases, a node does not even have knowledge of its neighboring nodes. This makes it difficult to direct a transmission over a secure channel to an intended recipient; the recipient node may have moved out of range or may never have been a neighbor.

In our CPAL-ES encoding of SRP, we have made some assumptions about the ad hoc network topology. The topological assumptions are for modeling purposes and are in no way intended to limit the scope of SRP. Rather, the assumptions are intended to make analysis more controlled while remaining faithful to the authors' design. For the purposes of our analysis, we assume the ad hoc network topology of Figure 4.1 with nodes `S`, `A`, `B`, and `T`. A link between two nodes represents the fact that the two nodes are within transmission range (i.e. assuming bi-directionality, packets can be sent to and received from each node).

SRP's full CPAL-ES encoding is given in Appendix A. We do not yet consider broadcast transmissions; we will address this issue later during our attack in Chapter 5 and the solution in Chapter 6. As such, all sends are issued securely (`=>`). Of particular interest in SRP's encoding is the security goal at the last line of the protocol, reproduced below.

```
S: assert(reverse_route' == found_route);
```

The assertion checks whether the route accumulated along the route reply equals the route field certified by `T`. While there may be intermediate goals of SRP, this is its primary goal

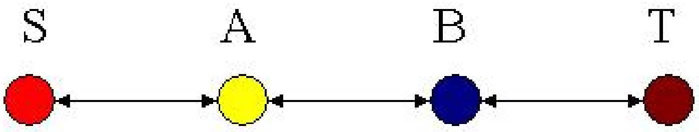

Figure 4.1. Ad hoc network topology.

and the one upon which we will focus. Stated simply, the goal of SRP is to identify a route between `S` and `T` that indeed exists. We now provide an analysis of how well SRP achieves this goal.

4.4 CPAL-ES Analysis of SRP

Appendix B shows the output of the run of SRP's encoding through CPAL-ES. At the very bottom, CPAL-ES produces the following predicate.

```
(<<<S.S,A.A>,B.B>,S.T> == <S.S,<A.A,<B.B,S.T>>>)
```

This identifies the weakest precondition that must be true before protocol execution for the security goal (assertion) to be achieved. The contents of the two concatenations show that this equality condition concerns the accumulated route. A cursory look at the predicate would have the casual observer believe this to be a rather trivial concession and perhaps an obvious assumption to make. However, we point out two important subtleties.

First, the order of concatenation is important. On the left-hand side we see that values are appended to the accumulated route, which occurs only during forward query propagation of the route request. Values on the right-hand side are prepended, which occurs only during the reverse route reply. While this may appear to be an annoying idiosyncrasy of CPAL-ES, it illustrates the importance of even a seemingly small detail such as the ordering of concatenation and the impact it can have on security. In fact, our analysis may not have identified the flaw had it not been for CPAL-ES's rigidness.

Second, and more importantly, notice that the contents of the concatenations (the fields) contain fully qualified variables in `A` and `B`'s address spaces. Recall, `A` and `B` are untrusted intermediate nodes. However, for the identified route to be verified by the source it must allow intermediate nodes to have a high level of involvement and participation in that route's establishment. But the very nature of `S`'s relationship with `A` and `B` indicates that it places little trust in these intermediate nodes. There is no jurisdiction over these nodes, so an entity cannot verify that they appropriately follow SRP's query propagation rules. Expressed another way, no entity can ensure that intermediate nodes append an IP address let alone their own. We show that this brings about the attack we introduce in the next chapter.

CHAPTER 5

ATTACK ON SRP

We first showed in [15] that an attack on SRP is possible by a single malicious intermediate node without collusion. This attack exposes a fundamental weakness of broadcast communications, namely that without some authentication scheme it is difficult to detect precisely **who** sent a message.

The attack arises when a malicious intermediate node `I` does not append its IP address to the *route* field of the SRP header. Recall, target node `T` uses the accumulated route to establish a path between `S` and itself. Even when `I` does not append its IP address, according to SRP, `T` will verify the query packet and send a response packet along the reverse path.

5.1 The Attack

We use the ad hoc network represented in Figure 5.1 to illustrate the attack. We use CPAL-ES notation to provide the steps of the attack. Recall, the nodes operate in a broadcast environment so the notion `A` "sends to" `B` is not well defined. For our purposes, `A => B` implies `A` broadcasts a message intended for `B`; the receive statement (`<-`) determines which principal, in fact, receives the message. We are mostly concerned with the *route* field for our attack, so we ignore the other SRP fields. Accordingly, our attack only manipulates the accumulated route.

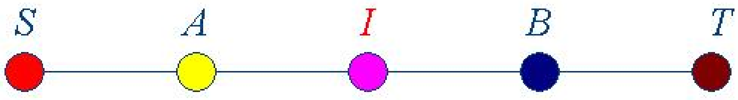

Figure 5.1. Attack network topology.

Below we show the actions during the route request propagation that lead to the attack.

```
 1. S: route := S;
 2. S: => A(route);
 3. A: <- route;
 4. A: => B(<route,A>);
 5. I: <- route;
 6. I: => B(route);
 7. B: <- route;
 8. B: => T(<route,B>);
 9. T: <- route;
10. T: route := <route,T>;
```

Notice, in steps (5) and (6) `I` intercepts the broadcast and simply forwards the identical packet onto `B` without appending its IP address. Still, `T` verifies the MAC and, because no fields used to compute the MAC have been altered, accepts the route request. `T` creates the route reply packet and computes a new MAC with the accumulated (false) route. The route reply follows this process.

```
 1. T: rev_route := T;
 2. T: => B(route,rev_route);
 3. B: <- (route, rev_route);
 4. B: => A(route,<rev_route,B>);
 5. I: <- (route,rev_route);
 6. I: => A(route,rev_route);
 7. A: <- (route,rev_route);
 8. A: => S(route,<rev_route,A>);
 9. S: <- (route,rev_route);
10. S: rev_route := <rev_route,S>;
```

Again, notice that `I` does not append its IP address. Once `S` receives the route reply it verifies the MAC and then tests `route` and `rev_route` for equality. This check is satisfied so

S accepts the route reply and deems the *route* SABT legitimate. Therefore, I has successfully deceived S and thwarted the security goals of SRP.

5.2 Implications

The result of the attack is that S erroneously believes a route exists between T and itself that does not depend on the participation of intermediate node I. Yet, S undoubtedly relies on I to forward packets so they finally reach T.

To illustrate a possible implication of S's false belief concerning the route, consider what happens when I leaves the ad hoc network. Any route maintenance technique will be unable to notify S that the route is no longer intact because it is believed the route does not rely on I's presence. If one extrapolates this, it is easy to envisage that the intruder could become involved in many such routes and thus have some control over the ad hoc network.

Consequences aside, the attack shows that SRP is flawed. SRP does not meet its goal to guarantee that the identified route is non-corrupted. The fundamental characteristic of this attack is that untrusted intermediate nodes can forward a packet without appending their IP addresses. The reliance of the source and destination on the actions of untrusted intermediate nodes produces this vulnerability.

5.3 CPAL-ES Analysis of SRP Attack

To encode the attack, we utilized the insecure send (->). Recall, this requires transmitted messages to go through malicious intruder I regardless of the true intended recipient. We provide the essence of our attack in the following snippet; the full encoding of the attack can be found in Appendix C.

```
A: <- (route);
A: route := <route,B>;
A: -> B(route);
I: <- (route);
I: => B(route);
B: <- (route);
```

This shows A receiving the *route* field and appropriately appending its identifier. A insecurely transmits the route to B, which malicious intruder I intercepts. I does **not** append its identifier, but simply forwards the message onto B. This leaves little doubt that the identified route relies on I's participation, and yet I is not included in the accumulated *route* field. Not only does B not possess a mechanism to detect the malicious activity, but worse still target node T will verify the route request it later receives. T subsequently accepts the route and creates a route reply with the route as an integral part. The reply follows the steps outlined in Section 5.1. Upon receipt, S will accept the route with no knowledge of I.

When we ran the encoding of SRP found in Appendix C through CPAL-ES, we formally verified our hypothesis. The predicate CPAL-ES produces (see Appendix D) shows what we already knew from Section 4.4. There is no indication of I's malicious activity or even that the protocol had been exploited.

CPAL-ES analysis confirms our attack. It provides further insight by showing that untrusted intermediate nodes may perform any of a number of malicious activities and thwart detection. While the MACs produced by S and T limit the extent of such activity, this security measure is inadequate to mitigate malicious activity and resulting compromises to security.

CHAPTER 6

SOLUTION

In [2] the authors introduce *watchdog* as a means to detect and mitigate node misbehavior. We first proposed an adaptation to *watchdog* we called *bloodhound* in a discussion panel [15]. *bloodhound* was designed as an extension to SRP and it detects the attack of Chapter 5. First, we shall describe *watchdog* and then our *bloodhound* adaptation.

6.1 Watchdog

As in Figure 5.1, suppose a path exists between `S` and `T` with intermediate nodes `A`, `I`, and `B`. Recall that because `A` cannot reach `B` directly, it routes packets through `I` who is within transmission range of both `A` and `B`. When `I` forwards the packet, not only does `B` receive it but `A` also overhears it. The ability of `A` to overhear this transmission is a result of the bi-directionality assumption we introduced in Chapter 2. In this way, `A` can determine if its packet has been forwarded properly.

The authors make use of *watchdog* to determine if a packet identical to the one `A` sent is forwarded by `I`. To this end, `A` maintains a buffer of recently transmitted packets. If `A` overhears a packet matching one in this buffer, then `A` knows `I` has properly handled the forward and the buffer entry is erased. On the other hand, if a specified timeout period elapses and `A` does not overhear the forwarding of its packet then `A` increments a failure tally kept for `I`. Once the tally exceeds a threshold, `I`'s behavior is flagged as malicious.

6.2 Bloodhound

We first consider the adaptation that solves the root cause. Upon further analysis, we showed in [16] that this solution injects a subtle vulnerability. We provide our full solution in Section 6.2.2.

6.2.1 Closing the SRP gap

Initially, the adaptation to *watchdog* was minor: the scheme's steps remained identical while only the interpretation of and response to a buffer match differed. Because *bloodhound* is an add-on to SRP, we know how a "loyal" intermediate node behaves (see Section 2.2). In short, an intermediate node need only append its IP address to the *route* field then forward the packet. As a result, A should **never** overhear a packet identical to one it has previously sent.

To illustrate, let's revisit our example from Chapter 5. Suppose A forwards a query packet with $route = \mathtt{SA}$. Any intermediate node $\mathtt{N}_i$ within A's transmission range will receive the packet, append its IP address ($\mathtt{N}_i$) and forward the packet. A overhears this forward and notes that the packet with $route = \mathtt{SAN}_i$ is **not** identical to one it sent, signaling that the packet has been properly forwarded.

Suppose, however, that the intermediate node is malicious ($\mathtt{N}_i = \mathtt{I}$) and that it attempts to exercise the attack from Chapter 5. In this case, I will not append its IP address to the route. Instead, I forwards exactly what it receives from A. A will overhear the packet and notice that the packet is identical to one it transmitted. When A encounters a match with an entry in its transmission buffer it knows malicious activity is being perpetrated by one of its neighbors. However, A does not have any way of discerning which node in its transmission range is acting maliciously. In this event, A can choose from two courses of action. First, if S and A have a SA and an established route, then A can notify S that a route reply with a given Q_{ID} and A as a route member is suspect. The action is then left up to S. Second, A can flag the entry in its buffer "malicious." If A later receives a response packet with the same Q_{ID} as a "malicious" entry, then it drops (does not forward) the response. In this way, S will never receive a corrupted route.

The second approach seems the more elegant of the two: it is simple, it does not presume the existence of a SA between S and A, and it requires no decision on the part of S. However, it injects error into the system in the form of a denial-of-service (DoS) attack. An example should help to clarify. Suppose we have the network topology of Figure 6.1. In this attack, I is **not** on the route between S and T. However, I is within transmission range of A and, thus, overhears the packet A forwards. Suspecting that *bloodhound* is in use, I can simply forward

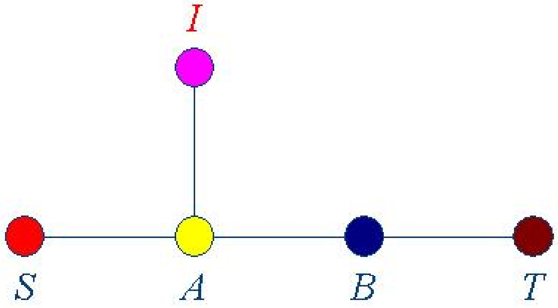

Figure 6.1. DoS attack network topology.

an identical packet to the one it received from A. Due to bi-directionality, A overhears this packet and labels its entry in the transmission buffer “malicious.” As a result, any response packet A receives with a matching Q_{ID} is dropped. I has successfully employed a DoS attack.

6.2.2 DoS removal

The error occurs because nodes running *bloodhound* over-react, per se. To resolve this problem intermediate nodes must react more cautiously. Expanding upon our example from Section 6.2.1, when A overhears a packet identical to one it transmitted, instead of labeling its entry in the transmission buffer “malicious” we recommend labeling it “suspicious.” As Figure 6.1 shows, A also overhears a packet B forwards with *route* = SAB. When A receives a response packet from T it can check to see if the *route* field is T*B where * represents one or more intermediate nodes. Recall, A can identify the route response by Q_{ID} and the accumulated route in the response is in reverse order. If the route matches A’s expectation then A can forward the response instead of dropping it as the simple solution of Section 6.2.1 suggests.

More formally, let TR_A denote the set of nodes within transmission range of A. For all v $\in TR_A$, if A overhears a packet with route S*Av, A will make note in its buffer that it could receive a response with *route* = T*v. When A receives such a response it appends its IP address to the *route* field and forwards the reply. A then flushes its buffer of all entries with the response packet’s Q_{ID}. If, however, A does not receive a response for the request with

Q_{ID}, it flushes its buffer of those entries after some specified timeout period elapses as in *watchdog*.

6.3 CPAL-ES Analysis of *bloodhound*

The *bloodhound* solution was our first encoding that required broadcast transmissions. For *bloodhound* to work, nodes must be able to overhear packets their neighbors transmit. Because CPAL-ES has no mechanism that specifically supports broadcast, we needed to address this issue with this work. We found a way to simulate the broadcast paradigm, a snippet of which we provide below; the full encoding of SRP with *bloodhound* is given in Appendix E.

```
1. A: -> B(msg);
2. I: <- (msg);
3. I: => A(msg);
4. A: <- (overheard_msg);
5. I: => B(msg);
6. B: <- (msg);
```

First, notice that to simulate broadcast transmissions a principal must issue multiple send statements. This is what `I` does in steps (3) and (5). These send statements assume some knowledge of the network topology, but no more than we have previously assumed for our modeling purposes. Once a node overhears a message, it executes *bloodhound*. Because of our topology assumptions, nodes that are in a position to overhear only execute *bloodhound*. In practice a node would possess the code to execute SRP and *bloodhound* and perform conditional checks to determine which block of code to execute. However, for simplicity and to keep the encoding short, we chose to execute only the *bloodhound* code on those nodes that overhear broadcasts.

The CPAL-ES encoding of *bloodhound* uses assertion statements to compare the overheard message to the one it transmitted. If there is a match, the route request identified by Q_{ID} is flagged as suspicious. Due to expressive limitations of CPAL-ES we discuss in Chapter 8, we were not able to set such a flag – we also had difficulty encoding the DoS removal scheme of Section 6.2.2. However, we did develop a scheme that identifies the erroneous route in the CPAL-ES analysis. By recognizing identical forwarded packets, a node executing *bloodhound* is able to send bogus Q_{ID} and Q_{seq} values in its route reply forward to `S`. These bogus values convince the source node to drop the response and thus the identified route it contains. When the source asserts the security goal, CPAL-ES requires these bogus values to equal specific values for a true result. This demonstrates that the bogus values serve their purpose. The output of CPAL-ES can be found in Appendix F.

CHAPTER 7

COMMENTS ON SRP'S APPROACH

The "leap-frogging" approach used by SRP to accumulate relevant route information using untrusted intermediate nodes proved to be flawed. This, coupled with SRP's non-colluding assumption, make the protocol suspect for use in truly hostile environments. It is unrealistic and impractical to imagine an environment without the potential for colluding entities. While less critical ad hoc network applications may adhere to these constraints, a battlefield environment with highly sophisticated adversaries should not be bound by such limitations. A DoS attack in such a scenario could prove disastrous.

The simple possibility of having even two colluding nodes renders the *bloodhound* solution to SRP ineffectual. We illustrate this with an example. The aim of the protocol is for `S` to establish a route to communicate with `T`, where `T` can be reached through `P`, `Q`, and `B` with `P` and `Q` being malicious colluding nodes. Figure 7.1 represents this situation. Each node in the network has SRP and *bloodhound* codes available. As in Chapter 5, we are only concerned with the *route* field. The series of events for a colluding attack are given in CPAL-ES notation.

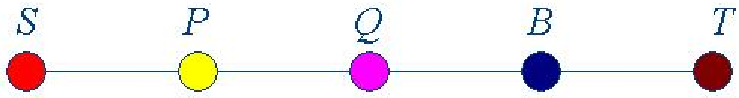

Figure 7.1. DoS attack with collusion.

```
S: route := S;
S: => P (route);
P: <- route;
```

```
P: => Q (<route,P>);
Q: <- route;
Q: => B (route);
B: <- route;
B: => T (<route,B>);
T: <- route;
T: => route := <route,T>;
```

Notice that Q does not append its IP address. Because P colludes with Q, it will not report such misbehavior detected by *bloodhound* to S. T is unaware of any discrepancy that might have occurred en route. As such, the response generated by T contains the route SPBT within the MAC. The sequence of events the route reply undergoes on its way from T to S are as follows.

```
T: route := T;
T: => Q (route);
B: <- route;
B: => P (<route,B>);
Q: <- route;
Q: => P (<route,Q>);
P: <- route;
P: => S (<T,B,P>);
S: <- route;
S: => route := <route,S>;
```

S accepts the route to T without any knowledge of the malicious activity. Therefore, P and Q have effectively deceived S into believing that it has a valid path to T which is not dependent on Q. This proves hazardous for S when it wishes to communicate with T, because the identified route is undoubtedly dependent on both P and Q.

The attack above uses adjacent colluding nodes. However, no location constraints are placed on the colluding nodes as the next example demonstrates. A similar attack occurs when colluding nodes are more than one link apart. In this scenario the amount of computation is exactly the same as that in the case when colluding nodes are adjacent; the difference occurs in the values malicious nodes alter in the route request and route response packets. In Figure 7.2, P and Q are malicious colluding nodes. When S sends a route request to T, the packet first reaches P. The packet is forwarded correctly according to SRP (IP addresses are appended to the *route* field) until the packet reaches Q.

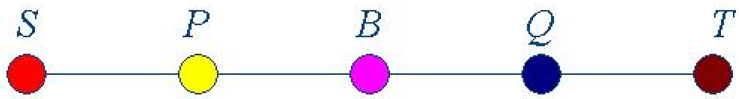

Figure 7.2. DoS with non-adjacent colluding nodes.

Q appends its address, but also deletes B's address from the *route* field. Q then forwards the packet to T. In processing the route response, Q follows SRP by appending its IP address, whereas P now removes B's entry from the *route* field before appending its own address. So we have removed a legitimate node, B, from the route without detection.

A property of both *bloodhound* and *watchdog* allows such malicious activity even in their proper use. Recall that the solution calls for intermediate nodes to compare outgoing traffic against previously transmitted packets. For example, when S generates a route request it compares the packet it transmitted with the packet P transmits. An alarm is raised **only if** the packets are identical. So when P does not append its address to the packet, *bloodhound* detects the discrepancy and triggers an alarm. However, in the example above the forwarded packets are not **identical** (their *route* fields differ), despite being incorrect. So even *bloodhound* is thwarted with clever attacks.

CHAPTER 8

FUTURE WORK WITH CPAL-ES

An important part of any research effort is to answer the question “What next?” The answer requires not only a breadth of background knowledge, but also evolutionary foresight. In this chapter we discuss those features of CPAL-ES that facilitate further wireless security protocol investigation; we also point out short-comings and offer suggestions that will benefit future researchers.

8.1 Features of CPAL-ES

As we discussed in the introduction to CPAL-ES in Chapter 4, the primary contribution of CPAL-ES is that it requires assumptions to be stated explicitly. This defends against an all too common pitfall of using a security protocol in unintended ways. We also discussed the idiosyncratic enforcement CPAL-ES has on the ordering of concatenations. Indeed, this helped to identify the fundamental flaw in SRP.

Here we will focus on an initially surprising feature of encryption that, upon further thought, stands to reason. In our initial encoding of SRP, we used encryption to produce the MAC for both the route request and the route reply. While we later replaced encryption with the more accurate global hash function, we felt an encryption performed by two principals who share a symmetric key would equate equally. However, this is not the case. Let us demonstrate this with the following example.

```
global: assume(S.kst == T.kst);
S: => T(msg);
T: <- (msg);
S: mac := e[msg]kst;
T: mac := e[msg]kst;
```

```
global: gassert(S.mac == T.mac);
```

Here we assume that S and T share a symmetric key. S then sends a message to T who duly receives it. The two principals perform an encryption of the same variable under their shared key. The assertion then compares the resulting outputs of the encryptions.

Initially it seemed surprising that CPAL-ES did not reduce the assertion to true. Instead, it required the following predicate to be true.

```
(e[S.msg]T.kst == e[S.msg]S.kst)
```

Aside from a simple substitution, the predicate is exactly what we asserted. Though surprising, when one considers the possibility of probabilistic encryption [24, 25] the catch by CPAL-ES makes sense. With probabilistic encryption, the deterministic results of encryption are removed; stated more clearly, an encryption can take on one of any number of values at different times. This reduces the amount of information leakage caused by cipher-text disclosure and makes cipher-text-only attacks by intruders that much more difficult. Under its current implementation, CPAL-ES supports the concept of probabilistic encryption.

8.2 Suggested Changes to CPAL-ES

As we have already discussed, our work needed to address ways to formally model the broadcast paradigm. Because CPAL-ES does not support a broadcast semantic, we simulated broadcasts with multiple sends (see Section 6.3).

However, we suggest an extension to the syntax of CPAL-ES to better facilitate study of wireless security protocols. We introduce the concept of a broadcast domain (BD) that encompasses all nodes within transmission range. Instead of simulating a broadcast with multiple sends, Alice instead initializes a group BD and issues only one send as follows.

```
A: BD := <B,C,E>;
A: => BD(msg);
B: <- (B.msg);
C: <- (C.msg);
E: <- (E.msg);
```

Also, we must consider use of secure send with broadcast. Does broadcast stay true to the semantics of a secure send? Surely, in this environment an intruder can intercept broadcast transmissions; this interpretation leans more toward our definition of an insecure send. Furthermore, we may want to consider extending the grammar to allow an optional interception by principal I for insecure sends.

We noticed in our encoding that there are times we would have liked to compare a variable field to the principal identifier. For example, instead of assuming the network topology we could have all code run on each node. In this way, a node could test to determine if it is the destination node and act appropriately. CPAL-ES currently provides no such mechanism.

Lastly, in our analysis we noticed the current working version of CPAL-ES does not support the full grammar expressed in [26]. In particular, logical operators such as AND, OR, and NOT are not supported, and neither is assignment of constants to variables. This limits the expressiveness of the notation and may even prevent accurate analysis of a number of wireless protocols. As a final aside, we would have liked to compare our SRP analyses with those of existing protocols but found the expressive limitations of CPAL-ES hindered our ability to do so.

CHAPTER 9

CONCLUSIONS

We introduced the Secure Routing Protocol with the goal to identify non-corrupted routes in mobile ad hoc networks. We showed that the two approaches the authors use to guard against malicious intermediate activity (see Section 2.5) are insufficient. We exposed a vulnerability with attack from Chapter 5.

This attack was found through intuitive reasoning and theoretical approaches. We discussed in Chapter 3 BAN logic and its inability to find the flaw in SRP. We further argued that BAN is not intended for proofs of security; we employed a more rigorous approach using the formal methods of CPAL-ES. We demonstrated that formal methods can be used in the broadcast environment, but have suggested that other models should be considered or existing models should be adapted to better suit the environment.

In Chapter 6 we discuss our solution *bloodhound* that mitigates untrusted node misbehavior. We also comment on SRP's unrealistic assumption of non-collusion. We consequently remove the assumption and expose inherent vulnerabilities that SRP and protocols designed in its likeness share.

Finally, we would like to say that this paper in no way is intended as a condemnation of secure routing in mobile ad hoc networks. Rather, we hoped to emphasize the difficulty in proposing security services in such an environment. Furthermore, we respect the attempts of previous researchers and hope that our analysis of the techniques used in the design of SRP contributes to this field of study.

APPENDIX A

CPAL-ES ENCODING OF SRP

```
--
-- initial assumptions
--
global: assume(S.kst == T.kst);
--
--Route request
--
S: mac := hash(S,T,Qid,Qseq,kst);
S: forward_route := S;
S: msg := <S,T,Qid,Qseq,forward_route,mac>;
S: => A(msg);
--
--Route request propagation
--
A: <- (msg');
A: (S,T,Qid,Qseq,forward_route,mac) := msg';
A: forward_route' := <forward_route,A>;
A: msg := <S,T,Qid,Qseq,forward_route',mac>;
A: => B(msg);

B: <- (msg');
B: (S,T,Qid,Qseq,forward_route,mac) := msg';
B: forward_route' := <forward_route,B>;
```

```
B: msg := <S,T,Qid,Qseq,forward_route',mac>;
B: => T(msg);
--
--Route request receipt
--
T: <- (msg');
T: (S,T,Qid,Qseq,forward_route,mac') := msg';
T: mac := hash(S,T,Qid,Qseq,kst);
T: found_route := <forward_route,T>;
T: assert(mac == mac');
T: newmac := hash(S,T,Qid,Qseq,found_route,kst);
--
--Route reply
--
T: reverse_route := T;
T: msg := <S,T,Qid,Qseq,found_route,reverse_route,newmac>;
T: => B(msg);
--
--Route reply propagation
--
B: <- (msg');
B: (S,T,Qid,Qseq,found_route,reverse_route,mac) := msg';
B: reverse_route' := <B,reverse_route>;
B: msg := <S,T,Qid,Qseq,found_route,reverse_route',mac>;
B: => A(msg);

A: <- (msg');
A: (S,T,Qid,Qseq,found_route,reverse_route,mac) := msg';
A: reverse_route' := <A,reverse_route>;
A: msg := <S,T,Qid,Qseq,found_route,reverse_route',mac>;
A: => S(msg);
```

```
--
--Route reply receipt
--
S: <- (msg’);
S: (S,T,Qid,Qseq,found_route,reverse_route,mac’) := msg’;
S: mac := hash(S,T,Qid,Qseq,found_route,kst);
S: assert(mac == mac’);
S: reverse_route’ := <S,reverse_route>;
--
-- SRP’s security goal
--
S: assert(reverse_route’ == found_route);
```

APPENDIX B

CPAL-ES ANALYSIS OF SRP

```
global: assume((T.kst == S.kst));
S: S.mac := f.hash(S.S,S.T,S.Qid,S.Qseq,S.kst);
S: S.forward_route := S.S;
S: S.msg := <S.S,S.T,S.Qid,S.Qseq,S.forward_route,S.mac>;
S: => A(S.msg);
A: <-(A.msg’);
A: (A.S,A.T,A.Qid,A.Qseq,A.forward_route,A.mac) := A.msg’;
A: A.forward_route’ := <A.forward_route,A.A>;
A: A.msg := <A.S,A.T,A.Qid,A.Qseq,A.forward_route’,A.mac>;
A: => B(A.msg);
B: <-(B.msg’);
B: (B.S,B.T,B.Qid,B.Qseq,B.forward_route,B.mac) := B.msg’;
B: B.forward_route’ := <B.forward_route,B.B>;
B: B.msg := <B.S,B.T,B.Qid,B.Qseq,B.forward_route’,B.mac>;
B: => T(B.msg);
T: <-(T.msg’);
T: (T.S,T.T,T.Qid,T.Qseq,T.forward_route,T.mac’) := T.msg’;
T: T.mac := f.hash(T.S,T.T,T.Qid,T.Qseq,T.kst);
T: T.found_route := <T.forward_route,T.T>;
T: assert((T.mac’ == T.mac));
T: T.newmac := f.hash(T.S,T.T,T.Qid,T.Qseq,T.found_route,T.kst);
T: T.reverse_route := T.T;
T: T.msg := <T.S,T.T,T.Qid,T.Qseq,T.found_route,T.reverse_route,T.newmac>;
```

```
T: => B(T.msg);
B: <-(B.msg');
B: (B.S,B.T,B.Qid,B.Qseq,B.found_route,B.reverse_route,B.mac) := B.msg';
B: B.reverse_route' := <B.B,B.reverse_route>;
B: B.msg := <B.S,B.T,B.Qid,B.Qseq,B.found_route,B.reverse_route',B.mac>;
B: => A(B.msg);
A: <-(A.msg');
A: (A.S,A.T,A.Qid,A.Qseq,A.found_route,A.reverse_route,A.mac) := A.msg';
A: A.reverse_route' := <A.A,A.reverse_route>;
A: A.msg := <A.S,A.T,A.Qid,A.Qseq,A.found_route,A.reverse_route',A.mac>;
A: => S(A.msg);
S: <-(S.msg');
S: (S.S,S.T,S.Qid,S.Qseq,S.found_route,S.reverse_route,S.mac') := S.msg';
S: S.mac := f.hash(S.S,S.T,S.Qid,S.Qseq,S.found_route,S.kst);
S: assert((S.mac' == S.mac));
S: S.reverse_route' := <S.S,S.reverse_route>;
S: assert((S.found_route == S.reverse_route'));
 *** End of Protocol ***
((<<<S.S,A.A>,B.B>,S.T> == <S.S,<A.A,<B.B,S.T>>>)
 or
 not (TRUE))
****** Simplified predicate follows.
(<<<S.S,A.A>,B.B>,S.T> == <S.S,<A.A,<B.B,S.T>>>)
```

APPENDIX C

CPAL-ES ENCODING OF SRP ATTACK

```
--
--initializations
--
global: assume(S.kst == T.kst);
--
--Route request
--
S: mac := hash(S,T,Qid,Qseq,kst);
S: forward_route := S;
S: msg := <S,T,Qid,Qseq,forward_route,mac>;
S: => A(msg);
--
--Route request propagation
--
A: <- (msg’);
A: (S,T,Qid,Qseq,forward_route,mac) := msg’;
A: gassert(A.Qid == S.Qid);
A: forward_route’ := <forward_route,A>;
A: msg := <S,T,Qid,Qseq,forward_route’,mac>;
--
-- insecure send
--
A: -> B(msg);
```

```
--
-- malicious intermediate node simply forwards to B
--
I: <- (msg);
I: => B(msg);

B: <- (msg’);
B: (S,T,Qid,Qseq,forward_route,mac) := msg’;
B: gassert(B.Qid == S.Qid);
B: forward_route’ := <forward_route,B>;
B: msg := <S,T,Qid,Qseq,forward_route’,mac>;
B: => T(msg);
--
--Route request receipt
--
T: <- (msg’);
T: (S,T,Qid,Qseq,forward_route,mac’) := msg’;
T: gassert(T.Qseq == S.Qseq);
T: mac := hash(S,T,Qid,Qseq,kst);
T: found_route := <forward_route,T>;
T: assert(mac == mac’);
T: newmac := hash(S,T,Qid,Qseq,found_route,kst);
--
--Route reply
--
T: reverse_route := T;
T: msg := <S,T,Qid,Qseq,found_route,reverse_route,newmac>;
T: => B(msg);
--
--Route reply propagation
--
```

```
B: <- (msg');
B: (S,T,Qid',Qseq,found_route,reverse_route,mac) := msg';
B: assert(Qid == Qid');
B: reverse_route' := <B,reverse_route>;
B: msg := <S,T,Qid,Qseq,found_route,reverse_route',mac>;
B: => A(msg);

A: <- (msg');
A: (S,T,Qid',Qseq,found_route,reverse_route,mac) := msg';
A: assert(Qid == Qid');
A: reverse_route' := <A,reverse_route>;
A: msg := <S,T,Qid,Qseq,found_route,reverse_route',mac>;
A: => S(msg);
--
--Route reply receipt
--
S: <- (msg');
S: (S,T,Qid',Qseq',found_route,reverse_route,mac') := msg';
S: assert(Qid == Qid');
S: assert(Qseq == Qseq');
S: mac := hash(S,T,Qid,Qseq,found_route,kst);
S: assert(mac == mac');
S: reverse_route' := <S,reverse_route>;
S: assert(reverse_route' == found_route);
```

APPENDIX D

CPAL-ES ANALYSIS OF ATTACK

```
global: assume((T.kst == S.kst));
S: S.mac := f.hash(S.S,S.T,S.Qid,S.Qseq,S.kst);
S: S.forward_route := S.S;
S: S.msg := <S.S,S.T,S.Qid,S.Qseq,S.forward_route,S.mac>;
S: => A(S.msg);
A: <-(A.msg’);
A: (A.S,A.T,A.Qid,A.Qseq,A.forward_route,A.mac) := A.msg’;
A: gassert((S.Qid == A.Qid));
A: A.forward_route’ := <A.forward_route,A.A>;
A: A.msg := <A.S,A.T,A.Qid,A.Qseq,A.forward_route’,A.mac>;
A: -> B(A.msg);
I: <-(I.msg);
I: => B(I.msg);
B: <-(B.msg’);
B: (B.S,B.T,B.Qid,B.Qseq,B.forward_route,B.mac) := B.msg’;
B: gassert((S.Qid == B.Qid));
B: B.forward_route’ := <B.forward_route,B.B>;
B: B.msg := <B.S,B.T,B.Qid,B.Qseq,B.forward_route’,B.mac>;
B: => T(B.msg);
T: <-(T.msg’);
T: (T.S,T.T,T.Qid,T.Qseq,T.forward_route,T.mac’) := T.msg’;
T: gassert((S.Qseq == T.Qseq));
T: T.mac := f.hash(T.S,T.T,T.Qid,T.Qseq,T.kst);
```

```
T: T.found_route := <T.forward_route,T.T>;
T: assert((T.mac’ == T.mac));
T: T.newmac := f.hash(T.S,T.T,T.Qid,T.Qseq,T.found_route,T.kst);
T: T.reverse_route := T.T;
T: T.msg := <T.S,T.T,T.Qid,T.Qseq,T.found_route,T.reverse_route,T.newmac>;
T: => B(T.msg);
B: <-(B.msg’);
B: (B.S,B.T,B.Qid’,B.Qseq,B.found_route,B.reverse_route,B.mac) := B.msg’;
B: assert((B.Qid’ == B.Qid));
B: B.reverse_route’ := <B.B,B.reverse_route>;
B: B.msg := <B.S,B.T,B.Qid,B.Qseq,B.found_route,B.reverse_route’,B.mac>;
B: => A(B.msg);
A: <-(A.msg’);
A: (A.S,A.T,A.Qid’,A.Qseq,A.found_route,A.reverse_route,A.mac) := A.msg’;
A: assert((A.Qid’ == A.Qid));
A: A.reverse_route’ := <A.A,A.reverse_route>;
A: A.msg := <A.S,A.T,A.Qid,A.Qseq,A.found_route,A.reverse_route’,A.mac>;
A: => S(A.msg);
S: <-(S.msg’);
S: (S.S,S.T,S.Qid’,S.Qseq’,S.found_route,S.reverse_route,S.mac’) := S.msg’;
S: assert((S.Qid’ == S.Qid));
S: assert((S.Qseq’ == S.Qseq));
S: S.mac := f.hash(S.S,S.T,S.Qid,S.Qseq,S.found_route,S.kst);
S: assert((S.mac’ == S.mac));
S: S.reverse_route’ := <S.S,S.reverse_route>;
S: assert((S.found_route == S.reverse_route’));
 *** End of Protocol ***
((<<<S.S,A.A>,B.B>,S.T> == <S.S,<A.A,<B.B,S.T>>>)
or
not (TRUE))
****** Simplified predicate follows.
```

```
(<<<S.S,A.A>,B.B>,S.T> == <S.S,<A.A,<B.B,S.T>>>)
```

APPENDIX E

CPAL-ES ENCODING OF *BLOODHOUND*

```
--
--initializations
--
global: assume(S.kst == T.kst);
--
--Route request
--
S: mac := hash(S,T,Qid,Qseq,kst);
S: forward_route := S;
S: msg := <S,T,Qid,Qseq,forward_route,mac>;
S: => A(msg);
--
--Route request propagation
--
A: <- (msg’);
A: (S,T,Qid,Qseq,forward_route,mac) := msg’;
A: gassert(A.Qid == S.Qid);
A: forward_route’ := <forward_route,A>;
A: msg := <S,T,Qid,Qseq,forward_route’,mac>;
--
-- simulating broadcast, send back to source
--
A: => S(msg);
```

```
--
-- S overhears
--
S: <- (overheard_msg);
--
-- insecure send
--
A: -> B(msg);
--
-- malicious intermediate node simply forwards to B
--
I: <- (msg);
I: => A(msg);
--
-- A overhears
--
A: <- (overheard_msg);
--
-- bloodhound
--
A: (S',T',Qid',Qseq',overheard_route,mac') := overheard_msg;
--
-- assures it's the same packet
--
A: assert(S == S');
A: assert(T == T');
A: assert(Qid == Qid');
A: assert(Qseq == Qseq');
A: assert(mac == mac');
--
-- the "kicker", should not happen if SRP packet forwarded properly
```

```
--
A: assert(forward_route’ == overheard_route);
--
-- then forwards onto B
--
I: => B(msg);

B: <- (msg’);
B: (S,T,Qid,Qseq,forward_route,mac) := msg’;
B: gassert(B.Qid == S.Qid);
B: forward_route’ := <forward_route,B>;
B: msg := <S,T,Qid,Qseq,forward_route’,mac>;
B: => T(msg);
--
--Route request receipt
--
T: <- (msg’);
T: (S,T,Qid,Qseq,forward_route,mac’) := msg’;
T: gassert(T.Qseq == S.Qseq);
T: mac := hash(S,T,Qid,Qseq,kst);
T: found_route := <forward_route,T>;
T: assert(mac == mac’);
T: newmac := hash(S,T,Qid,Qseq,found_route,kst);
--
--Route reply
-- presume reply is unicast along identified route
--
T: reverse_route := T;
T: msg := <S,T,Qid,Qseq,found_route,reverse_route,newmac>;
T: => B(msg);
--
```

```
--Route reply propagation
--
B: <- (msg');
B: (S,T,Qid',Qseq,found_route,reverse_route,mac) := msg';
B: assert(Qid == Qid');
B: reverse_route' := <B,reverse_route>;
B: msg := <S,T,Qid,Qseq,found_route,reverse_route',mac>;
--
-- Insecure send
--
B: -> A(msg);
--
-- Malicious intermediate node forwards
--
I: <- (msg);
I: => A(msg);

A: <- (msg');
A: (S,T,Qid',Qseq,found_route,reverse_route,mac) := msg';
A: assert(Qid == Qid');
--
-- added for bloodhound
-- will emulate a dropped reply with bad data check
--
A: if(forward_route' == overheard_route) then {
   msg:= <S,T,bad_Qid,bad_Qseq,found_route,reverse_route',mac>;
   } else {
   reverse_route' := <A,reverse_route>;
   msg := <S,T,Qid,Qseq,found_route,reverse_route',mac>;
   }
A: => S(msg);
```

```
--
--Route reply receipt
--
S: <- (msg’);
S: (S,T,Qid’,Qseq’,found_route,reverse_route,mac’) := msg’;
S: assert(Qid == Qid’);
S: assert(Qseq == Qseq’);
S: mac := hash(S,T,Qid,Qseq,found_route,kst);
S: assert(mac == mac’);
S: reverse_route’ := <S,reverse_route>;
S: assert(reverse_route’ == found_route);
```

APPENDIX F

CPAL-ES ANALYSIS OF *BLOODHOUND*

```
global: assume((T.kst == S.kst));
S: S.mac := f.hash(S.S,S.T,S.Qid,S.Qseq,S.kst);
S: S.forward_route := S.S;
S: S.msg := <S.S,S.T,S.Qid,S.Qseq,S.forward_route,S.mac>;
S: => A(S.msg);
A: <-(A.msg’);
A: (A.S,A.T,A.Qid,A.Qseq,A.forward_route,A.mac) := A.msg’;
A: gassert((S.Qid == A.Qid));
A: A.forward_route’ := <A.forward_route,A.A>;
A: A.msg := <A.S,A.T,A.Qid,A.Qseq,A.forward_route’,A.mac>;
A: => S(A.msg);
S: <-(S.overheard_msg);
A: -> B(A.msg);
I: <-(I.msg);
I: => A(I.msg);
A: <-(A.overheard_msg);
A: (A.S’,A.T’,A.Qid’,A.Qseq’,A.overheard_route,A.mac’) := A.overheard_msg;
A: assert((A.S’ == A.S));
A: assert((A.T’ == A.T));
A: assert((A.Qid’ == A.Qid));
A: assert((A.Qseq’ == A.Qseq));
A: assert((A.mac’ == A.mac));
A: assert((A.overheard_route == A.forward_route’));
```

```
I: => B(I.msg);
B: <-(B.msg');
B: (B.S,B.T,B.Qid,B.Qseq,B.forward_route,B.mac) := B.msg';
B: gassert((S.Qid == B.Qid));
B: B.forward_route' := <B.forward_route,B.B>;
B: B.msg := <B.S,B.T,B.Qid,B.Qseq,B.forward_route',B.mac>;
B: => T(B.msg);
T: <-(T.msg');
T: (T.S,T.T,T.Qid,T.Qseq,T.forward_route,T.mac') := T.msg';
T: gassert((S.Qseq == T.Qseq));
T: T.mac := f.hash(T.S,T.T,T.Qid,T.Qseq,T.kst);
T: T.found_route := <T.forward_route,T.T>;
T: assert((T.mac' == T.mac));
T: T.newmac := f.hash(T.S,T.T,T.Qid,T.Qseq,T.found_route,T.kst);
T: T.reverse_route := T.T;
T: T.msg := <T.S,T.T,T.Qid,T.Qseq,T.found_route,T.reverse_route,T.newmac>;
T: => B(T.msg);
B: <-(B.msg');
B: (B.S,B.T,B.Qid',B.Qseq,B.found_route,B.reverse_route,B.mac) := B.msg';
B: assert((B.Qid' == B.Qid));
B: B.reverse_route' := <B.B,B.reverse_route>;
B: B.msg := <B.S,B.T,B.Qid,B.Qseq,B.found_route,B.reverse_route',B.mac>;
B: -> A(B.msg);
I: <-(I.msg);
I: => A(I.msg);
A: <-(A.msg');
A: (A.S,A.T,A.Qid',A.Qseq,A.found_route,A.reverse_route,A.mac) := A.msg';
A: assert((A.Qid' == A.Qid));
A: if ((A.overheard_route == A.forward_route'))  then {
     A.msg := <A.S,A.T,A.bad_Qid,A.bad_Qseq,A.found_route,A.reverse_route',A.mac>;
   } else {
     A.reverse_route' := <A.A,A.reverse_route>;
   }
A: => S(A.msg);
S: <-(S.msg');
S: (S.S,S.T,S.Qid',S.Qseq',S.found_route,S.reverse_route,S.mac') := S.msg';
S: assert((S.Qid' == S.Qid));
S: assert((S.Qseq' == S.Qseq));
S: S.mac := f.hash(S.S,S.T,S.Qid,S.Qseq,S.found_route,S.kst);
S: assert((S.mac' == S.mac));
S: S.reverse_route' := <S.S,S.reverse_route>;
S: assert((S.found_route == S.reverse_route'));
 *** End of Protocol ***
((((<<<S.S,A.A>,B.B>,S.T> == <S.S,A.reverse_route'>)
and
(A.bad_Qseq == S.Qseq))
and
(A.bad_Qid == S.Qid))
or
not (TRUE))
****** Simplified predicate follows.
(((<<<S.S,A.A>,B.B>,S.T> == <S.S,A.reverse_route'>)
and
(A.bad_Qseq == S.Qseq))
and
(A.bad_Qid == S.Qid))
```

REFERENCES

[1] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[2] Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. *Mobile Computing and Networking*, pages 255–65, 2000.

[3] E. Royer and C-K. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications Magazine*, pages 46–55, April 1999.

[4] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad-hoc networks. *IEEE Network*, November 2001.

[5] Stephen Carter and Alec Yasinsac. Secure position aided ad hoc routing protocol. In *Proc. of the LASTED International Conference on Communications and Computer Networks (CCN02)*, pages 329–34, Nov 4-7, 2002.

[6] Kimaya Sanzgiri, Bridget Dahill, Brian N. Levine, and Elizabeth M. Belding-Royer. A secure routing protocol for ad hoc networks. In *International Conference on Network Protocols (ICNP), Paris, France*, November 2002.

[7] S. Yi, P. Naldurg, and R. Kravets. Security-aware ad-hoc routing for wireless networks. Technical Report UIUCDCS-R-2001-2241, University of Illinois at Urbana-Champaign, August 2001.

[8] P. Papadimitratos and Z. J. Haas. Secure routing for mobile ad hoc networks. *SCS Communication Networks and Distributed Systems (CNDS 2002)*, January 27-31 2002.

[9] Douglas R. Stinson. *Cryptography: Theory and Practice.* Chapman & Hall/CRC, 2nd edition, 2002.

[10] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, Inc., 2nd edition, 1996.

[11] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[12] Z.J. Haas. A new routing protocol for the reconfigurable wireless networks, 1997.

[13] Z.J. Haas, M. Perlman, and P. Samar. The interzone routing protocol (ierp) for ad hoc networks, 2001.

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

[15] J. Marshall. An analysis of SRP for mobile ad hoc networks. In *Proc. of the 2002 International Conference on Wireless Networks (ICWN'02), Las Vegas, NV*. CSREA Press, July 2002.

[16] John Marshall, Vikram Thakur, and Alec Yasinsac. Identifying flaws in the secure routing protocol. In *Proc. of the International Performance Computing and Communications Conference (IPCCC'03), Phoenix, AZ*, April 9-11 2003.

[17] P. Bieber. A logic of communication in hostile environment. In *Proc. Computer Security Foundations Workshop III*. IEEE Computer Society Press, Los Alamitos, CA, June 1990.

[18] P. Syverson. The use of logic in the analysis of cryptographic protocols. In *Proc. of the 1991 IEEE Symposium on Research in Security and Privacy, Oakland, CA*. IEEE CS Press, Los Alamitos, CA, May 1991.

[19] P. Syverson. Knowledge, belief, and semantics in the analysis of cryptographic protocols. *Journal of Computer Security*, 1:317–334, 1992.

[20] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *ACM Operating Systems Review*, 21(1), January 1987.

[21] Alec Yasinsac and Wm. A. Wulf. A framework for a cryptographic protocol evaluation workbench. *The International Journal of Reliability, Quality and Safety Engineering (IJRQSE)*, 8(4):373–89, 1 December 2001.

[22] Justin Childs. Evaluating the TLS family of protocols with weakest precondition reasoning. Technical Report TR-000703, Florida State University, July 2000.

[23] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), October 1969.

[24] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental paker keeping secret all partial information. In *Proc. of the 14th ACM Symposium on the Theory of Computing*, pages 316–29, 1982.

[25] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–99, April 1984.

[26] Alec Yasinsac. *Evaluating Cryptographic Protocols*. PhD thesis, University of Virginia, 1996.

BIOGRAPHICAL SKETCH

John D. Marshall, II

After receiving his Bachelor of Science in Mathematics & Computer Science from Elon University, NC, John joined Florida State University's security group in the Fall of 2001 as a Harris Corporation Fellow. While at FSU, John has conducted research in the areas of wireless security, mobile ad hoc networks (MANETs), secure routing protocols and mobile agent security. John authored and co-authored papers analyzing secure routing protocols and presented his findings at conferences in Las Vegas and Phoenix.

In his spare time John enjoys playing guitar, contemplating the meaning of life, traveling, learning about things he doesn't understand, playing Ultimate, meeting fresh faces, driving, wearing his newly acquired cowboy hat, reading, and chasing that pesky little white ball around the links.