FORMAL SECURITY ANALYSIS OF A SECURE ON-DEMAND ROUTING PROTOCOL FOR AD HOC NETWORKS USING MODEL CHECKING

by

Evren Önem

B.S. in Mathematics, Boğaziçi University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2007

DEDICATION

To the best warrior I have ever met; my father...

ACKNOWLEDGEMENTS

I'd like to thank most to my father for he has taught me well what I need to survive, and for he has shown me there is no thing such luck but endeavor. Most of this thesis is written while watching him at distressful nights where he has been fighting with the insidious disease inside; and though he could not see that I have completed this thesis, I am sure he can somehow feel it is now done...

Secondly comes my mother for the endless patience, love and trust she has inside for her son.

While standing in the mist with a blank mind, obviously I could have a hard time searching for the light without the vast experience of my thesis supervisor, Ufuk Çağlayan.

I specially offer my deepest thanks to A. Burak Gürdağ who has never withheld his mind from covering my deficiencies throughout all phases of this work.

I am also truly grateful to H. Birkan Yılmaz for all his effort to make me feel restful and confident throughout the writing phase of this thesis.

Lastly, I must express my gratitude to Özlem Çetin who has abode me ever since she has known me, especially through all my distressed graduate years.

This work is supported by The State Planning Organization under "The Next Generation Satellite Networks and Applications" project, No: DPT 2003-K120250.

ABSTRACT

FORMAL SECURITY ANALYSIS OF A SECURE ON-DEMAND ROUTING PROTOCOL FOR AD HOC NETWORKS USING MODEL CHECKING

An ad hoc network is a self-configuring network of mobile terminals, connected by wireless links and exhibiting nomadic behavior by freely moving within an area. Computing the routes between the terminals in the ad hoc environment and delivering a guarantee of communication have never been achieved by any protocol in its entirety. In this work, we model an ad hoc network to model-check ARIADNE in order to verify one of its powerful security properties. By a similar approach to Buttyán's *Active-1-2* attack on ARIADNE, we have used SPIN to flag a sequence of possible events in the protocol leading to a new *Active-2-2* attack, where two compromised nodes collaborate to remove all intermediate nodes from the route-discovery process.

ÖZET

KABLOSUZ TASARSIZ AĞLAR İÇİN GÜVENLİ BİR YÖNLENDİRME PROTOKOLÜNÜN MODEL KONTROL TEKNİĞİ İLE FORMAL GÜVENLİK İNCELEMESİ

Kablosuz tasarsız ağ dediğimiz kavram, belli bir alanda serbestçe hareket ederek gezgin bir davranış tarzı sergileyen, kablosuz olarak birbirine bağlı hareketli terminallerden oluşan bir bilgisayar ağıdır. Kablosuz tasarsız ağlardaki terminaller arasında yönlerin hesaplanması ve iletişim garantisinin sağlanabilmesi şu ana dek hiçbir protokol tarafından tam manasıyla sağlanamamıştır. Bu çalışma, bir tasarsız ağ yönlendirme protokolü olan ARIADNE'nin güçlü bir güvenlik özelliğinin doğrulanması amacıyla formal olarak modellenmesi ve model-kontrolünün yapılmasını içermektedir. Buttyán'in ARIADNE üzerinde sergilenebileceğini gösterdiği *Active-1-2* tipi saldırıya benzer bir yaklaşımla, SPIN yazılımı ARIADNE protokolünde *Active-2-2* tipinde yeni bir saldırıya yol açan olası bir haberleşme serisi bulması için kullanıldı. Bu saldırıda paketin gittiği yol üzerindeki iki anlaşmalı terminal, yol keşfetme işlemi sırasında aralarındaki tüm diğer terminalleri yokmuş gibi gösterebilmektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABR	Associativity Based Routing Protocol
AODV	Ad hoc On-demand Distance Vector
ARA	Ant-Colony Based Routing Algorithm for MANETs
AtSe	Attack Searcher
AVISPA	Automated Validation of Internet Security Protocols and Applications
BCY	Beller-Chang-Yacobi Protocol
BGP	Border Gateway Protocol
CBRP	Cluster-Based Routing Protocol
CGSR	Cluster-head Gateway Switch Routing
CL	Computational-Logic
CPAL-ES	Cryptographic Protocol Analysis Language Evaluation System
CSP	Communicating Sequential Processes
CTL	Computation Tree Logic
DDR	Distributed Dynamic Routing
DoS	Denial-of-Service
DPLL	DPLL/Davis-Putnam-Logemann-Loveland algorithm
DSDV	Destination-Sequenced Distance Vector
DSR	Dynamic Source Routing
DST	Delay Sensitive Transport
ECFSM	Extended Communicating Finite State Machine
FDR	Failures Divergence Refinement
FDT	Formal Description Techniques
FORP	Flow-Oriented Routing Protocol
FSR	Fish-eye State Routing
GPSAL	GPS Ant-Like routing algorithm
GSR	Global State Routing
HLPSL	High Level Protocol Specification Language

HOL	Higher Order Logic Theorem Prover
HSR	Hierarchical State Routing
ITU-T	International Telecommunication Union-Telecommunication Stardardization Sector
KDC	Key Distribution Center
LAR	Location-Aided Routing
LMR	Lightweight Mobile Routing
LOTOS	Language Of Temporal Ordering Specifications
LTL	Linear Temporal Logic
MAC	Message Authentication Code
MANET	Mobile Ad hoc Network
MMWN	Mobile Multimedia Wireless Networks
MSC	Message Sequence Chart
NIST	National Institute of Standards and Technology
NRL	Naval Research Laboratory
OFMC	On-the-Fly Model-Checker
OLSR	Optimized Link State Routing
OSRP	On-demand Secure Routing Protocol
PAN	Protocol Analyzer
PROMELA	Process Meta Language
RDMAR	Relative Distance Microdiscovery Ad hoc Routing
RERR	Route-Error
ROAM	Routing On-demand Acyclic Multi-path
RREP	Route-Reply
RREQ	Route-Request
S-AODV	Secure Ad hoc On-demand Distance Vector
SAR	Security-aware Ad hoc Routing
SATMC	SAT-based Model-Checker
SDL	Specification and Description Language
S-DSDV	Secure Destination-Sequenced Distance Vector
SEAD	Secure Efficient Ad hoc Distance vector
SLSP	Secure Link-State Protocol

SLURP	Scalable Location Update-based Routing Protocol
SMT	Secure Message Transmission
SMV	Symbolic Model Verifier
SPIN	Simple Promela Interpreter
SRP	Secure Routing Protocol
SSA	Signal Stability-based Adaptive routing
STAR	Source Tree Adaptive Routing
SUCV	Statistically Unique and Cryptographically Verifiable
TBRPF	Topology dissemination Based on Reverse-Path Forwarding
TLA	Temporal Logic of Actions
TLS	Transport Layer Security
TORA	Temporally Ordered Routing Algorithm
WRP	Wireless Routing Protocol
DREAM	Distance Routing Effect Algorithm for Mobility
ZHLS	Zone-Based Hierarchical Link State Routing Protocol
ZRP	Zone Routing Protocol

1. INTRODUCTION

An ad hoc network is a self-configuring network of mobile terminals, connected by wireless links and exhibiting nomadic behavior by freely moving within an area; the union of which form an arbitrary topology. Such networks are distinguished by their rapidly and unpredictably changing presence or absence of links. Following the widespread use of PDA's, laptops and 802.11/WiFi technologies, concerns on secure communication in mobile ad hoc networks have gained significance in the research area.

Computing the routes between the terminals in mobile ad hoc networks and delivering a guarantee of communication success have never been achieved in its entirety. The proposed secure routing protocols mostly come with an informal security analysis performed by their designers, which makes it doubtful to trust the protocol. It is now widely accepted that secure protocols need a proof of security before being accepted as secure. Here comes the main flavor of formal methods, which are defined to be mathematical techniques for specification and verification of complex systems. Through the use of proper formal methods, it is possible to achieve provable correctness and reliability in any system design and to analyze a system for desired properties.

The main objective of this thesis is to apply a famous formal verification approach, model checking, on the secure routing protocol ARIADNE, which aims for resilience against multiple compromised nodes and arbitrary attackers in a mobile ad hoc network. With this work, model checking approach is used for the first time in the literature for the purpose of verifying a security property of an ad hoc routing protocol. Using a similar approach proposed by Buttyán *et al.* [1], our aim is to use model checking to find a sequence of possible events in ARIADNE leading to a new *Active-2-2* attack, where two compromised nodes collaborate to remove all intermediate nodes from the route-discovery process. The reason we have chosen ARIADNE is not only for the fact that it has very powerful security properties, but also its authors have introduced a new attacker classification with the term *Active-y-x*; where y stands for the number of terminals that the attacker has compromised and x stands for the

total number of terminals that the attacker owns within the network.

In the second chapter, firstly, we give an overview of routing in mobile networks with both general routing issues and secure routing issues. A broad overview of current secure routing protocols is presented as possible prevention mechanisms; with some other approaches as possible detection mechanisms. Secondly, we discuss formal methods for system specification and verification. We mention various specification methodologies and verification tools, along with a shallow introduction to our tool of choice: SPIN. We also present a comprehensive slice of the related work history on modeling and analysis of protocols using formal methods. Thirdly, we discuss a particular version of ARIADNE: ARIADNE with MACs. We describe the main protocol operations very briefly and give a security discussion about the protocol.

In the third chapter, we discuss our own work: model checking of ARIADNE. Firstly we give discussions about effective use of SPIN and our assumptions. Secondly, we present our modeling approach for various network primitives, communication, legitimate network nodes, and compromised network nodes. Then, our simulation phase, security property specification phase, and verification phase with its results are discussed.

Lastly in the fourth chapter, conclusions and future directions are presented respectively.

2. OVERVIEW OF ROUTING IN MOBILE AD HOC NETWORKS AND FORMAL METHODS

2.1. Overview of Routing in Mobile Ad Hoc Networks

2.1.1. What is a MANET?

A mobile ad hoc network (MANET) can be described as a set of mobile terminals that are connected by wireless links and exhibiting nomadic behavior by freely moving within an area, dynamically and frequently establishing and breaking up associations with other terminals, possibly without disruption of node-to-node communication. The terminals have the capability to dynamically discover routes to other nodes, along with the ability to maintain these routes. The network may rapidly and unpredictably change its topology, thus rapidly and unpredictably changing the presence or absence of links. Such a network can be rapidly deployed without relying on some pre-existing fixed network infrastructure, may operate in a standalone fashion or may also be connected to some larger network such as Internet [2].

2.1.2. Routing in MANETs

MANET routing protocols are commonly classified in three categories, namely, proactive, reactive and hybrid protocols [3].

In *proactive protocols*, the routing information is exchanged between the neighbor terminals and is kept in local tables to use later while making routing decisions. Such behavior may provide a minimum routing delay as a result of its readiness; but as the rate of change in topology increases, it becomes harder and harder for a terminal to be up-to-date so the performance degrades dramatically. Destination-Sequenced Distance Vector (DSDV) [4] can be said to be a representative proactive protocol for MANETs since most others are derivatives of DSDV. WRP [5], GSR [6], FSR [7], STAR [8],

DREAM [9], MMWN [10], CGSR [11], HSR [12], OLSR [13] and TBRPF [14] are also proactive routing protocols proposed for MANETs.

In *reactive protocols*, any routing information is queried only when a terminal needs it. This behavior may react later than proactive protocols while acquiring a route to some destination but greatly decreases the control-information exchange within the network which is unnecessary unless used by a terminal. For this class of protocols, Dynamic Source Routing (DSR) [15] is the representative one. AODV [16], LMR [17], ROAM [18], TORA [19], ABR [20], SSA [21], RDMAR [22], LAR [23], ARA [24], FORP [25] and CBRP [26] are some other reactive routing protocols proposed for MANETs.

In *hybrid approaches*, advantages of both classes of protocols above are subject to be used in specific cases. Using proactive routing for near terminals and reactive routing for distant ones may be an example for the hybrid approaches. ZRP [27], ZHLS [28], SLURP [29], DST [30] and DDR [31] are hybrid protocols proposed for MANETs.

Within the rapidly changing characteristics of a MANET topology, the reactive protocols are known to outperform the proactive ones with the exception of some certain cases [32, 33].

2.1.3. Attacks on Routing Protocols for MANETs

Attacks on MANET routing protocols are classified as *passive* and *active* attacks. A passive attacker which is only a threat for privacy, is an eavesdropper on the network who may perform [34];

- *Sniffing*, where the attacker listens and records the traffic in the network.
- *Traffic analysis*, where the attacker intercepts and examines messages to deduce information from communication patterns [35]. Typically, the information that can be inferred from the traffic increases directly proportional to the number of messages observed. By traffic analysis, various advantages may be obtained, such as *location disclosure*, in which the attacker may discover the location of a node,

or even the entire structure of the network.

- *Deliberate exposure*, where the attacker disclosures the traffic information he/she recorded to the other terminals who do not have permission to see it.

Classification of active attacks are done in various ways in the literature, considering various different point of views. We prefer to classify them in two categories both of which may be said to be instances of a denial-of-service (DoS) attack from an application-layer perspective [36, 37, 38, 39]:

- *Routing-disruption attacks*, where the attacker tries to disrupt the legitimate routes that packets in the network will follow and make them travel through unintended routes. By distributing forged false information in the network, various attacks can be realized:

  - *Route detours* may be created where the legitimate terminals are deceived to use suboptimal routes.
  - *Partitions* can be created where one set of nodes within the network can be prevented from reaching another set.
  - *Gratuitous detours*, where the attacker makes a route seem longer than it actually is, by adding non-existent terminals to the route-reply.
  - *Blackmailing*, where the attacker might cause a selected terminal to be added to the blacklists of other terminals and make that terminal not appear in any route.
  - *Rushing attack* may be performed in protocols using duplicate route-request suppression; where the attacker broadcasts route-requests in account of other terminals thereby causing the legitimate route-requests in the future to be ignored.

  Additionally,

  - *Wormholes* can be established where a pair of attackers have a private physical connection and use this connection to disrupt routing, such as, one

recording route-requests may forward them to the other to be rebroadcast thereby causing routes longer than one or two hops to be undiscovered.
  - *Tunneling attack* may be performed where two remote terminals collaborate to exchange legitimate messages of other terminals in an encapsulated form through existing message channels in order to show themselves as adjacent nodes. In this way they may achieve having certain traffic through them.

- *Resource-consumption attacks*, where the attacker tries to acquire access to any resource in the network for which he/she does not have enough privilege. This type of attack may target bandwidth, memory or computation power of terminals. Some examples are;

  - *Spoofing or Masquerading*, where the attacker attempts to identify itself as some other terminal, which in turn will open the gate to perform further attacks such as creating routing loops.
  - *Hijacking*, where the attacker takes control of an on-going communication and masquerades as one of the communicating node.
  - *Misclaiming*, where the attacker advertises its authorized control of some network resources in a way that is not intended by the authoritative network administrator [40]. Compromised, unauthorized or masquerading nodes may misclaim network resources.
  - *Sleep deprivation torture attack* may be performed in which the attacker attempts to drain batteries of some other node by constantly keeping it busy in various ways.

  Also, there are more attacks which must be classified under both of the above attack types, since they may target both routing-disruption and resource-consumption.

  - *Blackholes* can be created by attacker terminals who attract packets and then drop all of them.
  - *Grayholes* are the derivatives of blackholes where not all but some packets are selectively dropped.

- *Routing loops* may be created by an attacker thereby causing the packets to travel in cycles without reaching their intended destinations.
- *False route-errors* may be forged and disseminated in order to damage valid routes and initiate new route-discoveries within the network.

2.1.4. Secure Routing in MANETs

From the routing protocol point of view, there are two types of messages in an ad hoc network [41]: *routing messages* and *data messages*; both of which need totally different mechanisms to be secured. While any point-to-point security solution can provide *confidentiality* for data messages; a means of *differentiating between the legitimate nodes* and *per-hop authentication* are needed by intermediate terminals for securing the routing messages, and such a point-to-point mechanism alone cannot provide such a means. Thus, the active focus on securing routing in MANETs is generally on securing the routing messages. Besides, it should be strongly underlined that a *prevention* mechanism is destined to be flawed if it is not perfect, i.e., if there is a way to abuse it; thus for security in a wider sense, a proper *detection* mechanism is essential to react. One can find a very detailed survey in [42] about prevention and detection schemes at each distinct layer; but we will mostly focus on network-layer prevention and detection schemes.

Prevention Mechanisms: These mechanisms include key-management and secure routing protocols:

- *Key-Management*: One of the most fundamental problems of a secure routing environment is the key-setup phase, which means disseminating the authentic key information to the mobile terminals. While in the case of simultaneous deployment, sharing the private keys before deployment is the most generic solution; in cases requiring incremental deployment, such as a MANET, establishing trust and keys between each two terminals becomes hard. F. Stajano and R. Anderson have proposed the *resurrecting duckling model* [43] and D. Balfanz *et al.* have given a more general shape to this approach by offering the use of privileged side

channels [44]. S. Zhu *et al.* [45] described a secure protocol against a collusive attack by up to a certain number of compromised nodes, enabling any two nodes in the ad hoc network to establish a pairwise shared key on the fly, without requiring the use of an on-line key distribution center. Z. Haas and L. Zhu [46] described a distributed service in which the trust is divided into some number of shares using threshold cryptography and these shares are assigned to some predefined number of arbitrarily chosen nodes, called servers. Their mechanism can tolerate a certain number of compromised servers if a certain number of partial signatures are provided to compute a correct signature. G. Montenegro and C. Castelluccia have proposed *statistically unique cryptographically verifiable (SUCV) addresses* [47], that reduce the problem of distributing a list of *(node, public-key)* pairs to distributing a list of legitimate nodes. S. Capkun and J. P. Hubaux, assuming a source routing protocol, have proposed a mechanism to provide secure routing even with an incomplete set of security associations, provided that the percentage of security associations is sufficiently high [48]. Some other approaches can be examined in [49, 50, 51].

- *Secure Routing Protocols*: In order to cure the flaws of the general routing protocols defined in Section 2.1.3, different secure routing protocols are proposed each with the ability of resisting some classes of attackers. Yih-Chun Hu *et al.* [52] introduced an attacker classification with the term *Active-y-x*, where y stands for the number of terminals that the attacker has compromised and x stands for the total number of terminals that the attacker owns within the network. In order to examine the current state-of-the-art in routing protocols and their resistance-levels to known flaws, one should at least consider:

  - *Secure AODV (S-AODV)*: Two mechanisms are used to secure the AODV messages; digital signatures to authenticate the non-mutable fields of the messages, and hash chains to secure the hop count information. For the non-mutable information, authentication is performed in an end-to-end manner [41]. Attacks that have been prevented successfully are: "impersonating some other terminal by forging a RREQ or a RREP", "creating a black-

hole for the subnet" and "forging a falsified RERR with a high destination sequence number in order to fail the future route discoveries of other terminals". But nothing prevents the attacker from passing on the received authenticator and hop-count without changing them. Tunneling and wormhole attacks may still be launched. Location disclosure is also possible.

- *Secure Efficient Ad hoc Distance vector (SEAD)*: Based on DSDV and uses one-way hash chains to prevent multiple uncoordinated attackers from creating incorrect routing state in any other terminal [53, 36]. A node uses a specific single next element from its hash chain in each routing update that it sends about itself. By the use of this initial element, the hash chain can provide authentication for the lower bound of the metric in other routing updates for this destination. Any routing update can be authenticated using one of the previous authentic hash values from the same hash chain. However, SEAD cannot prevent the attack where a terminal re-advertises the same advertisement for a particular *sequence number* (freshness) and *metric* (the node's shortest known distance). This is because SEAD only secures the lower bound on the metric ensuring that the terminal does not reduce it [54]. Blackhole, location disclosure and wormhole attacks can still be performed.
- *Authenticated Routing for Ad hoc Networks (ARAN)*: Provides authentication and integrity through the use of cryptographic certificates [55]. Also provides non-repudiation. It consists of two stages of which the second is optional for a trade-off between power-saving and security. ARAN is able to detect the attackers in the ad hoc environment; but as much as any mechanism using public key cryptography, it is vulnerable to DoS attacks launched by flooding the network with forged control messages for which signature verifications are required. Using this vulnerability, an attacker can force a terminal to discard a certain fraction of the control messages it receives [36]. Blackhole, location disclosure and wormhole attacks can still be performed.
- *Secure Routing Protocol (SRP)*: An extension to apply to the existing routing protocols so as to secure the route discovery phase [56], i.e., in order to

accept only the legitimate RREP for a RREQ through the use of message authentication codes. Only an end-to-end shared-key is required for the communicating pair of terminals. Flooding is limited since the neighbors are ranked inversely proportional to their send-rates and served accordingly. Since RERRs cannot be authenticated, a node on any route may forge a falsified RERR for that route. A node can freely modify the node-list of a RREQ packet that it forwards. SRP is also vulnerable to wormhole attack and attackers can at worst hide the routes they belong to [57]. An attacker may broadcast forged RREQs in the name of a legitimate terminal in order to reduce the effectiveness of the future legitimate RREQs of that terminal. A selfish node may not forward RREQs so that its own future RREQs will have higher priority. An attacker may also launch a masquerading attack using the security associations of the compromised terminal. Blackhole and location disclosure attacks can also be performed.

- *Secure Message Transmission (SMT)*: Different in the sense that it tries to ensure successful delivery of data packets forming an end-to-end security association, but not ever considering the security of route discovery and route maintenance phases [58]. Thus SMT should be deployed with some other protocol which can realize the route discovery phase and uses multiple routes each with a rating assigned by using feedback from the destination nodes. Messages are sent in a partial manner using secret sharing techniques so that if a certain number of the total packets are received, the message can be reconstructed at the destination. SMT handles link breakages and compromised routes by assigning them lower ratings.
- *Security-aware Ad hoc Routing (SAR)*: Introduces the notion of a *trust hierarchy* by distributing keys for each trust-level [59]. Each terminal has a certain immutable trust-level and while initiating a route discovery it declares the minimum value of a security metric (e.g. trust-level) of the terminals that will participate on the route; hence no node with questionable trust-level becomes a part of that route. The discovered route may not be optimal; but in terms of trust-levels, it is the most secure [54]. Scalability is a problem since it means distributing keys, but the protocol can prevent

the malicious nodes from being in the discovered route. Blackhole, location disclosure and wormhole attacks may still be performed.

- *Secure Link-State Protocol (SLSP)*: Uses digital signatures and one-way hash chains to secure the link-state updates which are signed and propagated a limited number of hops [60]. Terminals are assigned priorities inversely proportional to the number of link-state updates they generate or forward. So, the flooding attack described in SRP is also valid for SLSP. Blackhole, location disclosure and wormhole attacks may still be performed.
- *On-demand Secure Routing Protocol (OSRP)*: This protocol attempts to provide a fault-free path under situations where a group of nodes are possibly malicious, through assigning certain weights to each link between two adjacent nodes [61]. When a faulty link is found, its weight is multiplicatively increased so that the initiator of the route-request can avoid that link in the future by selecting from multiple routes the route whose sum of link-weights is the least, i.e., the route which has the least likelihood of having a faulty link inside. If there is one fault-free path to the desired destination, even in a highly adversarial environment, the protocol ensures the successful discovery after a bounded number of faults. Blackholes are prevented but grayholes are possible. Location disclosure and wormhole attacks may still be performed.
- *S-DSDV*: Relying on existing pair-wise secret-keys between each two terminals in the network, entity and message authentication is provided [62]. Data integrity protection and data origin authentication are realized. Besides, routes with falsified destination, advertised routes with falsified sequence numbers, advertised routes with falsified cost metrics, routing updates with falsified information, and advertised routes with falsified next hops are detected provided there is at most one bad node in the network. Blackhole, location disclosure and wormhole attacks may still be performed.
- *ARIADNE*: Yih-Chun Hu *et al.* [52] design this protocol to provide security against one compromised node and arbitrary active attackers. We will explain this protocol in detail in 2.3.

Some other secure routing approaches can be examined in [63, 64, 65, 66, 67, 68]. A much more detailed comparison between key-management schemes and secure routing protocols considering different metrics is provided in [69].

Detection Mechanisms: These mechanisms include *watchdog* and *pathrater* approaches [70], which target protection of the network from the misbehaving terminals by monitoring the neighbors for a future routing decision. Simulations show that, when used in combination, these two approaches provide considerable improvement in network throughput in an adversarial environment.

- *Watchdog*: Every terminal maintains a watchdog process monitoring their neighbors by listening promiscuously on the transmissions and checking whether the neighbor participates in forwarding process as expected. The misbehaviors are rated for each neighbor and after having exceeded a certain rating value the monitoring terminal notifies other nodes of the situation. The watchdog might not detect misbehaving nodes in presence of ambiguous collisions, receiver collisions, limited transmission power, false misbehavior, collusion between neighboring nodes and partial dropping [54]. Also the memory and computational resources are wasted for the actions performed while monitoring the neighbors, such as verifying the integrity of the sent packets.
- *Pathrater*: This approach enhances the knowledge of misbehaving nodes by considering also the link reliability data so as to pick the most reliable route. Terminals maintain a database storing link ratings of every other node. The optimal route is chosen considering an average link rating over the possible routes. If the watchdog reports a misbehaving terminal, its rating is set to a negative value; however, the recovery is possible for the malicious nodes since they can increment their rating after a period of healthy participation in routing.
- *Packet Leashes*: This approach targets the detection of a wormhole which may severely disrupt routing if remains uncovered. A leash is the information included in a packet to limit the maximum distance that the packet is allowed to be transmitted. Two different types of leashes are defined [71] both requiring different mechanisms; but both with the same purpose: to determine the max-

imum distance a packet may travel so that if a wormhole exists then it can be detected since the packets through the wormhole travel more distance than feasible. *Geographical leashes* need geographical location information to send with each packet, for the packet to be verified at the receiving terminal. This type requires the nodes to be aware of the maximum speed of a terminal and also requires loosely-synchronized clocks. *Temporal leashes* require the nodes to have tightly synchronized clocks so that each packet has a timestamp showing the time when it was sent. The receiving node can thereby conclude if the packet has traveled too far, by using the fact that the upper bound on the distance the packet can travel is determined by the speed of light. Both leashes suffer from either exact positioning information or accurate time synchronization both of which are difficult to achieve.

Some other detection approaches can be examined in [72, 73].

2.2. Overview of Formal Methods for System Specification and Verification

2.2.1. What are Formal Methods?

The term 'formal methods' is defined to be mathematical techniques for the specification, development and verification of software and hardware systems [74]. Formal methods specially focus on requirements and specification phases of development. The main flavor of formal methods is that it is possible to achieve provable correctness and reliability in any system design and to analyze a system for desired properties. Here, 'system' refers to any application whose behavior can be described in a formal language.

It is common for any informal system specification to lead to ambiguous definitions of the desired features; furthermore, there stand no reliable way to prove the completeness and consistency of the design. In such systems, informal inspection is prosecuted with error-checking test suites of which the coverage gets weaker directly proportional to the system complexity. Instead, for the system not to be totally re-

vised in case of any fault, mathematical techniques providing reliability and provability should be located. Formal methods have already demonstrated success in specifying commercial and safety-critical software, and in verifying protocol standards and hardware designs [75].

Formal methods may be used for some distinct purposes [76] such as formal specification, formal verification, rapid prototyping, functional testing and performance testing; of which only the first two are in our interest.

2.2.2. Formal Specification

Unambiguous determination of the system's main features and behaviors in a mathematical manner. The specification is meant to describe the 'what', not the 'how'. This formal description is generally used to guide further investigations, such as validation or verification. Recent advances have brought around the definition of formal description techniques (FDT), which can be classified according to their operational model as [76] finite state machine, process algebra, Petri Nets and timed automata. Since one of our main objectives is specifying a secure routing protocol, we will introduce some of the FDTs which are used in this field:

- *SDL (Specification and Description Language)*: A high-level description language supporting non-determinism. It is defined by ITU-T [77] and its focus is especially on telecommunication systems, process control and real-time applications [78]. SDL specifies systems as a set of interconnected abstract machines, which are actually extensions of finite state machines communicating on asynchronous channels by exchanging control messages. Processes have finite control states and change state with transitions triggered by message reception. SDL's nature, keeping structure and behavior apart, is very useful in describing protocol architectures. Verification of a complex system by SDL tools is performed by partial model checking the equivalent finite-state machine produced by the combination of control states and data values.*JADE* [79] and *SITE* [80] are public domain tools and *Telelogic Tau SDL Suite* and *Object-Geode* are commercial tools sup-

porting system development with SDL.

- *ESTELLE*: An international ISO standard that can model event-driven behavior and whose focus is especially on communication protocols [81]. It is based on an extended state-transition model supporting nondeterministic communication enhanced by the addition of Pascal language. ESTELLE can model a specified system as a hierarchical structure of communicating automata running in parallel and communicating by exchanging messages. An executable model can be generated from the formal specification. Pascal-support for data definitions and calculations makes Estelle suitable for implementation and testing issues rather than model checking.*Pet Dingo* is developed by NIST as a public domain Estelle tool, and *Estelle Development Toolset* is a commercial package by Institut National des Telecommunications.
- *LOTOS (Language Of Temporal Ordering Specifications)*: Standardized by ISO for specifying concurrent and communicating systems [82]. It is a language based on temporal ordering and uses process algebra as the modeling approach, which is an abstract language for formal specification and model concurrent communicating processes effectively through the use of logical operators. LOTOS specifies a system with a *declaration*, characterizing system behavior as composition of subsystems; and a *definition*, describing the behavior of each component within the system. Though its modeling and validation capabilities are very high thanks to its strong abstraction and support of model reduction, LOTOS friendliness is low due to its textual format. *Caesar/Aldebaran Development Package* provides efficient model checking, verification of logical temporal formulas and compilation of abstract data type libraries.
- *UPPAAL*: An instance of timed-automata operational model, which is a toolbox capable of specifying simulating and verifying systems that can be modeled as networks of timed automata extended with integer variables, structured data types, and channel synchronisation [83]. Its specification language is an extension of finite-state machines with clock variables. The query language used to specify properties to be checked is a subset of computation tree logic (CTL). Also, watchdog processes may be attached to the system, which are synchronized to detect undesired behaviors. The tool supports graphical display of specification,

simulation and validation phases.

- *AVISPA (Automated Validation of Internet Security Protocols and Applications)*: Aims at developing a push-button, industrial-strength technology for the analysis of large-scale Internet security-sensitive protocols and applications [84]. its High Level Protocol Specification Language (HLPSL) is based on Temporal Logic of Actions (TLA), which is itself a powerful language for specifying concurrent systems. Specification of a security protocol and its properties written in HLPSL are automatically translated to an Intermediate Format (IF) specification, which is an infinite-state transition system with an initial state, transition rules, and a state-based safety property defining whether a given state is an attack state or not. The IF specification is then sent to the back-ends, i.e., four different verification tools that can analyze IF specifications: an on-the-fly model-checker OFMC, a CL-based attack searcher AtSe, a SAT-based model-checker SATMC, and a tree automata-based protocol analyzer TA4SP. The tool's user-friendliness is at maximum since the validation process consists of a push-button action.
- *SPIN (Simple Promela INterpreter)*: This widely used tool will be explained in detail in Section 2.2.4.

A detailed comparison between FDTs considering various metrics, such as pertinence to protocols, modeling capabilities, validation capabilities, simulation capabilities, performance analysis, rapid-prototyping, tools availability, friendliness, main application domain and industrial applications, is given in [76].

2.2.3. Formal Verification

The act of proving or disproving the correctness of a system with respect to a certain formal specification or property [85], i.e., the possible behavior of the system is checked against the desired behavior. Two well-studied approaches are *model checking* and *theorem proving*.

- *Model Checking*: A technique that relies on building a finite model of a system and checking that a desired property holds in that model [86]. *Temporal model*

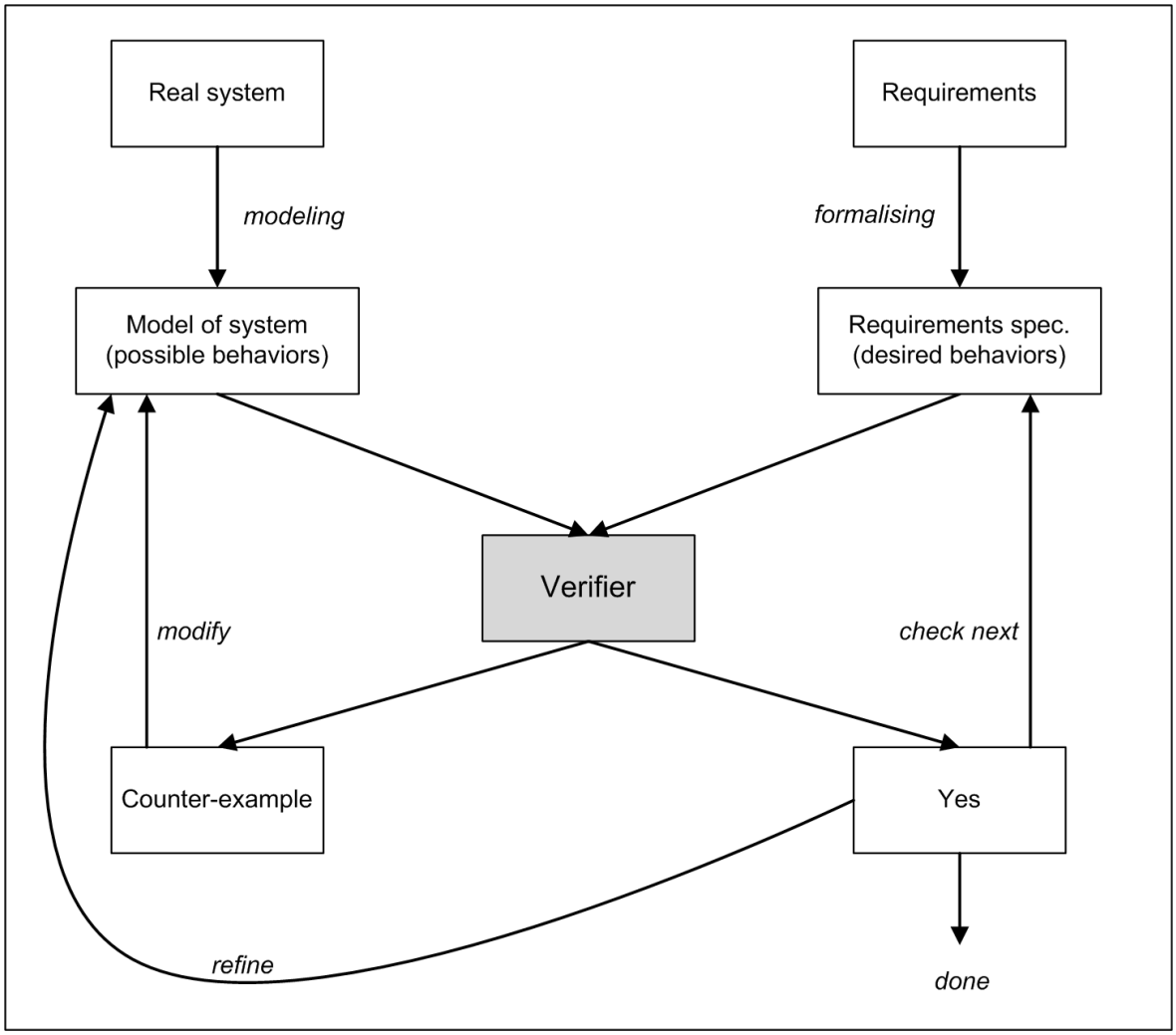

Figure 2.1. Verification methodology of model checking.

checking, in which the specifications are stated in a temporal logic and systems are modeled as finite state transition systems, is firstly developed by Clarke and Emerson [87] and by Queille and Sifakis [88] independently in the 80s. In a typical case, the user creates a high level representation for the system, i.e., the model, and the specification to be checked. The model-checker either terminates positively, meaning that the model satisfies the specification; or give an execution trace showing how the property in question might not hold (See Figure 2.1). The check is generally an exhaustive state space search, exploring all states and transitions in the model to see whether the required properties are valid in all the reachable states and the execution sequences of the model. Since the model is finite, this process is guaranteed to terminate. The size of the state space for such a model depends on the number of concurrent processes, the number and range

of the internal variables, the type of the exchanged messages, and the nature of the communication [76]. A model being more abstract means having less of the parameters above; so, it is usually much simpler to verify properties at a more abstract level. Main flavors of model checking are:

- It is fully automated and fast; in the sense that it takes as input a formally specified description of the system and a temporal logic formula to check the system against, and then automatically produce the answer without any user interaction.
- It is not needed to specify the whole system; model checking can also produce handy results on a partially defined system and accommodates making decisions about functionality in the early stages of development.
- It comes up with counter-examples i.e., error traces, which demonstrate the deficiency clearly, greatly helping to cure the error.
- Various case studies have shown that the use of model checking has led to shorter development times [89].

Main problems are:

- The state-explosion problem, which arises due to high complexity of the described system. To defeat this common problem, techniques like ordered *binary decision diagrams (BDDs)* [90] to represent state transition systems efficiently; *partial order reduction* [91] to reduce the number of independent interleavings of concurrent processes; *localization reduction* [92] to automatically abstract a model relative to the property being checked; *equivalence reduction of identifiers* [93] to collapse two distinct identifiers that are semantically equivalent; and *semantic minimization* [94] to eliminate unnecessary states from a system model, are developed. Even in the light of these approaches, the state-explosion problem may persist, for which the best solution is to find a more proper *abstraction* for the system under verification [95].
- In the cases where the system cannot be specified as a closed finite model,

model checking loses its suitability.

  - In communications protocol verification, model checking is generally run on a few concurrent processes (to the best of our knowledge, at most 5), due to state-explosion problem. Although it is possible to show under certain assumptions that this approach suffices to capture all possible behaviors of a protocol [96]; it may not be sufficient to generalize the verification results in some cases.
  - It is hard to choose what in the real world to model and to decide on the right level of abstraction. Models only 'model' some aspects of the system in question. The correspondence between the formal description and the real world is limited since the real world is not a formal system. So, a positive verification run for a system does not necessarily show that the system does not have deficiencies in the real world. But this is a very general modeling problem [75].

- *Theorem Proving*: A technique where both the system and its desired properties are expressed as formulas in some mathematical formalism, which defines a set of axioms and a set of inference rules for deduction of further facts from the given axioms. Theorem proving is the process of finding a proof of a property from the axioms of the system [86]. Depending on the formalism used, the hardness of proving a theorem may be in some spectrum from trivial to impossible. The theorem provers can be categorized depending on the degree of how automated they are; from highly automated general-purpose provers to interactive provers dedicated to some special purpose. Interactive provers are generally more suitable to prove some property of a system since they require user-guidance at certain phases of the operation; meaning that in order to use these provers efficiently, the user must have a degree of proficiency in the field. The distinctive feature of theorem provers is that they can naturally handle infinite state spaces, using numerous proof procedures (e.g. mathematical induction, model elimination, tableaux method, superposition, higher-order unification and DPLL) in order to generate proofs about infinite sets.

2.2.4. SPIN Model Checker

SPIN (Simple Promela INterpreter) is a widely accepted tool-box used for specification, simulation, validation and verification of asynchronous concurrent processes, such as the terminals in a communication protocol. It is used to extensively check high-level models of complex systems and to detect flaws in these system designs. The tool supports validation of logical consistency requirements, invariant assertions, and temporal properties expressed in *Linear Temporal Logic (LTL)* [97].

SPIN is based on *Extending Communicating Finite State Machine (ECFSM)* theory due to its structure consisting of a set of concurrent *processes*, extended with *variables* and data space, and communicate by exchanging structured messages through finite-length asynchronous *channels* [76]. SPIN's specification language is called *Promela (PROcess MEta LAnguage)*. The processes have deterministic and non-deterministic transition capabilities. Data objects are either global or process-local, and customized data structures can be defined. Channels are able to store messages containing any finite number of fields, and support several operations like polling, sorted send, random receive and synchronous rendez-vous communication. Atomic sequences, deterministic steps, selections, repetitions and escape sequences are the compound statements of PROMELA.

One can use SPIN in either one of two tightly-coupled operational modes: *simulation mode*, or *verification mode*. Simulation mode can be used to get an impression of the model and shape the further modeling; but not to prove any fact about the system. SPIN can generate optimized verifiers from the user-defined PROMELA model and then verification mode is used for checking any violations to the specified properties. In the case of a found counter-example, SPIN simulator displays an error-trace using guided simulation (See Figure 2.2).

SPIN relies on the fact that the combined execution of a system of asynchronous processes can be shown to be a product of automata, where each process is itself a finite-state automaton. A *Büchi automaton* is also built using the system's *never claim*. The

global system automaton is the synchronous product of this Büchi automaton with the asynchronous product of finite-state automata of the processes.

In distributed systems design, it is standard to make a distinction between two types of correctness requirements [95]: *safety* and *liveness*; where safety means the set of properties that the system may not violate and liveness means the set of properties that the system must satisfy. From verifier's angle, claims are also about either reachable or unreachable states, or feasible or infeasible execution sequences. Correctness claims in PROMELA can be expressed as *basic assertions*, *never claims*, or *trace assertions*; possibly through the use of *end-state labels*, *progress-state labels* and *accept-state labels*. Basic assertions constitute the only class of correctness claims that is allowed to be checked during simulation. All other classes need verification to be checked. Never claims, which can be automatically generated from LTL formulas using SPIN's built-in translator, provide a solid expressive power to specify the properties in inquisition. They define behavior which should *never* occur in any execution sequence of the system.

SPIN has various *compile-time options* to support different platforms and different types of use; *simulation options* to support customizing of the simulation runs of PROMELA models; *syntax checking options* to perform a syntactic correctness check of PROMELA models; *LTL conversion options* to support conversion from LTL formulae to never claims; and *model-checker generation options* to support fine-tuning of the verifier-code to a specific type of verification. The *protocol analyzer (PAN)* generated by SPIN has also options for applying partial order reduction, for increasing speed, and for reducing memory use.

2.2.5. Modeling and Analysis of Protocols Using Formal Methods

Several works have been done in the literature where formal methods are used for modeling and analysis of communication protocols. Mostly, the effort is on cryptographic security protocols; and any work on verification of security properties of MANET routing protocols can rarely be found.

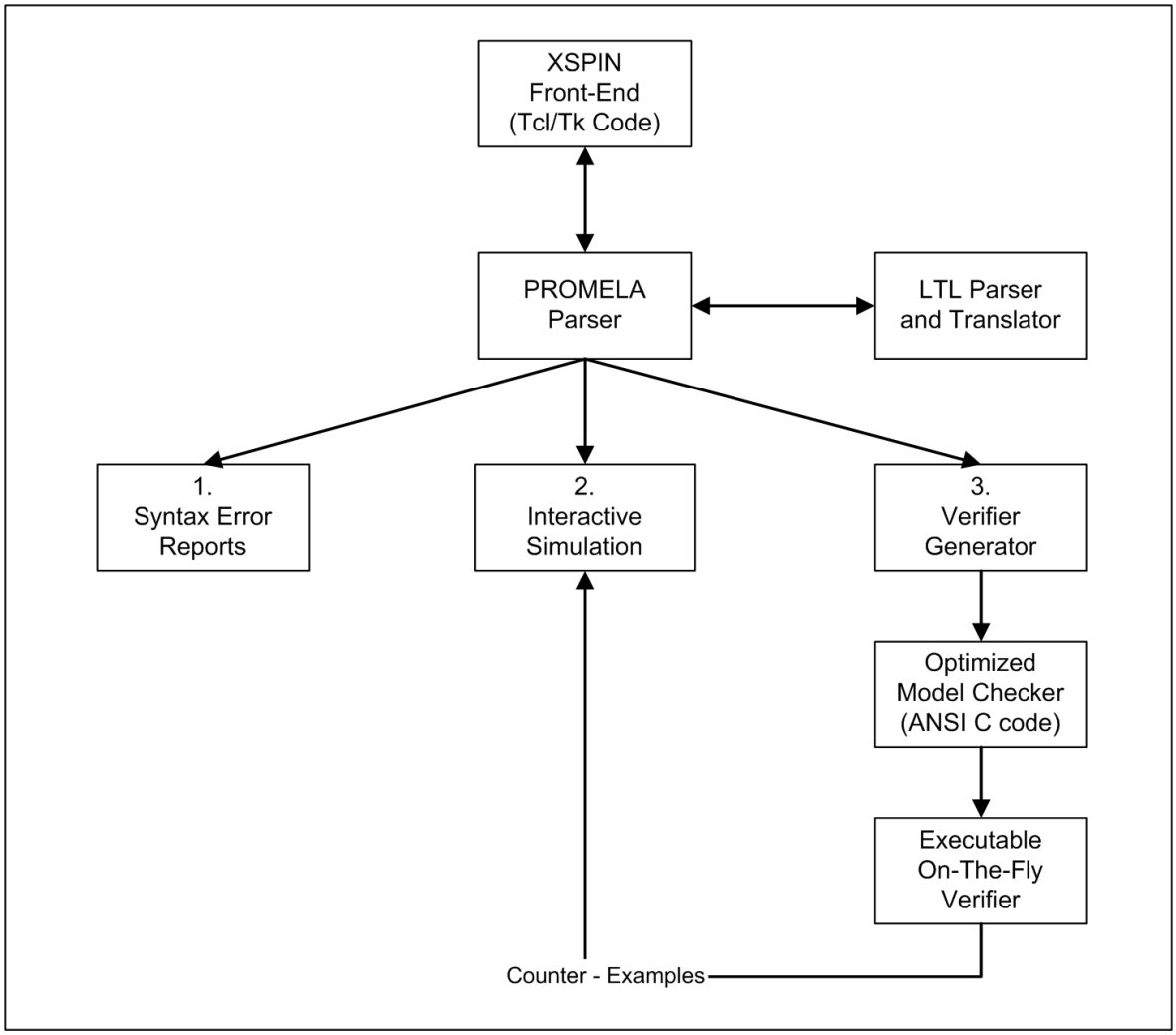

Figure 2.2. The Structure of SPIN simulation and verification.

Some of the *security protocol verifications* in the literature are:

- In [98], Lowe used a general-purpose model checking tool, FDR, to model security protocols, coming up with a previously unknown attack on the famous Needham-Schroeder Public Key protocol.

- In [99] and [100], Murφ is used by Mitchell *et al.* in the first place for analyzing some cryptographic protocols for identification of previously-known deficiencies; and then for identification of a previously-unknown flaw on Secure Sockets Layer (SSL) protocol.

- In [101], BAN logic that specifically targets formal analysis of authentication

protocols is introduced which is used later on to identify the flaws in the X.509 authentication protocol.

- In [102], the author demonstrated type-confusion attacks on the Group Domain of Interpretation Protocol with NRL protocol-analyzer.

- In [103], the authors proposed a fully-automatic verification tool, Athena, combining techniques from both model checking and theorem proving, tailored for security protocol analysis, with the capabilities of proving correctness with arbitrary number of concurrent runs.

- In [104, 105], the authors use Isabelle [106] to prove properties of the Kerberos authentication system and TLS protocol, respectively. In [107], the verification of Secure Electronic Transaction (SET), an e-commerce protocol by VISA and MasterCard, using the theorem prover Isabelle, is described. The results are important in the sense that enormous protocols such as SET are amenable to formal analysis. In [108], the authors specified a certified e-mail protocol, and by formalizing its correctness assertions, they verified the main guarantees of this protocol.

- In [109], Lowe *et al.* used Casper/FDR2 to successfully discover flaws in the Clark/Jacob library; it found previously known attacks on 20 protocols, and some previously unknown attacks on 10 other protocols.

- In [110], Chevalier and Vigneron present a new model for automated verification of security protocols, permitting the use of an unbounded number of protocol runs. The authors prove its correctness, completeness and give a termination guarantee. They demonstrate the implemented approach by rediscovering some known flaws and one previously unreported attack on Denning-Sacco symmetric key protocol.

- In [111], the authors examine Needham-Schroeder public-key authentication pro-

tocol, 1KP electronic commerce protocol [112], and Wide Mouthed Frog protocol; demonstrating some known flaws about these protocols using special-purpose model-checker Brutus.

- In [113], the process of formal verification using modal logics is discussed. The verification process is demonstrated by way of case studies on some security protocols: original BCY [114], and published modifications of BCY. An unknown flaw is identified; and a new proposal by the authors, resistant to this attack is verified.

- In [115], the authors model a signature system implemented on top of a secure operating system and validate access control and integrity properties using SPIN, by modeling a generic attacker who cannot only function as a legitimate user of the system, but can also call functions in unintended ways, with arbitrary parameters.

- In [116], Basin *et al.* present the on-the-fly model-checker OFMC, which is state-of-the-art both in terms of coverage and performance. Their demonstrations are impressive in the sense that the tool finds all known attacks and discovers a new one in a test-suite of 38 protocols from the Clark/Jacob library in a few seconds of CPU time for the entire suite.

- In [117], the authors describe the modelling of a two multicast group key management protocols in a first-order inductive model, and the discovery of previously unknown attacks on them by the automated inductive counter-example finder CORAL.

- In [118], the authors introduce a version of distributed temporal logic and thereby define a protocol-independent distributed communication model, on top of which protocols and security goals can be formalized and analyzed. They illustrate their approach by using a standard example: Needham-Schroeder Public Key Authentication Protocol.

- An instance of verification of a security protocol using SPIN is in [119], where

Maggi and Sisto analyzed Needham-Schroeder Public Key Authentication Protocol and discovered Lowe's attack [98]. They also verified the fixed version proposed in [98]. The authors compared their results in terms of the number of reachable states in the model, with the other approaches analyzing the same protocol using other model-checkers, Murφ and the CSP-based tools Casper and FDR. They described a procedure for the automatic generation of the intruder definition, using preliminary data-flow analysis to provide complexity reduction while building the model.

- For purposes of generic verification of security protocols, the authors in [120] formulate a "loss of secrecy" property and formalize a simplified attacker in PROMELA; then they prove that it is of equal power compared to Dolev-Yao intruder, in terms of analyzing secrecy properties.

Some work on *protocol verification for ad hoc networks* are:

- In [121], the authors applied formal verification on the GPSAL routing protocol and proved loop-freedom in a high frequency moving hops scenario. They also verified that the time of a message on the network is directly related to this moving.

- Another instance using SPIN is [122], where the authors performed an assisted verification on AODV using an interactive theorem prover, HOL, with the model-checker SPIN, in order to discover conditions causing the formation of routing loops.

- The most related work in the literature to ours is in [89], where the author verified Wireless Adaptive Routing Protocol (WARP) using SPIN against loop-freedom in any five nodes ad hoc network configuration.

- In [123], the author proved the convergence of the RIP protocol, provided a sharp real time bound for it; and proved that the protocol finds optimal routes. Further-

more, he identified flaws in earlier versions of AODV causing loops to be formed and suggested modifications, with which he proved the protocol to be loop-free. In addition, by proving a general theorem about an upper bound on convergence time, he derived important convergence properties for different classes of configurations in BGP.

- In [124], SRP is analyzed using both the canonical BAN logic analysis for cryptographic protocols and the CPAL-ES formal methods suite.

- In [125], a methodology based on CSP/FDR to validate critical properties of ad hoc networks is presented. Furthermore, they demonstrated the applicability of their methodology in a case study on Cluster-Based Routing Protocol.

- In [126], Liu demonstrated an appropriate approach to use SMV for verifying AODV protocol by describing models separately for each role and for each possible network topology.

- In [127], the authors considered the effect of the protocol parameters on the timing behaviour of AODV, through the use of UPPAAL. They highlighted a dependency of the lifetime of routes on network size, which can be alleviated by allowing the route timeouts to adapt to network growth.

- In [128, 129], Nanz and Hankin presented a broadcast process calculus with local storage operations to model the core of a distance vector protocol for ad hoc networks. They illustrated their approach by showing that under certain conditions a routing loop attack may be performed.

- In [130], the authors give a formal framework for the security analysis of on-demand source routing protocols for wireless ad hoc networks. The novelty of their approach is the application of the simulation-based approach, which has been proposed to prove the security of cryptographic protocols, in the context of ad hoc routing.

2.3. Overview of ARIADNE Protocol

2.3.1. What is ARIADNE?

ARIADNE [52] is an ad hoc network routing protocol that aims for resilience against *Active-y-x* attackers, relying only on efficient symmetric cryptography. The protocol operates in an on-demand fashion, based on the well-known DSR protocol; meaning that the terminals dynamically discover routes to the related destination only when needed. Other than the generic approach of enhancing an existing insecure protocol by equipping it with cryptography, the authors redesigned each routing primitive and its processing from scratch. The protocol offers three different modes of operation; in the sense that authentication of routing messages can be realized by one of three proposed schemes: shared secrets between each pair of nodes (ARIADNE with MACs), shared secrets between communicating nodes combined with broadcast authentication (ARIADNE with TESLA), or digital signatures. Our focus is on ARIADNE with MACs; and when we mention about 'ARIADNE', what we refer to is 'ARIADNE with MACs' from here on.

2.3.2. Assumptions and Design Goals

The authors' assumptions about the environment are:

- Physical layer and MAC layer attacks are disregarded, since these problems have various other standalone solutions that must be kept separate from the designer's point of view.
- The network may reorder, replay, corrupt, or even lose packets while transmitting.
- The protocol assumes bidirectional links.
- The needed assumptions for the operation mode of the protocol are inherited. For instance, a key distribution center (KDC), or some other means of setting up $n(n+1)/2$ shared secrets in a network with n nodes is assumed for ARIADNE. Authors offer some alternative mechanisms to achieve this.
- A terminal trusts the terminal with which it communicates.

The authors' design goals are:

- preventing attackers or compromised nodes from tampering with uncompromised routes consisting of uncompromised nodes;
- resilience against Active-y-x attackers through simple and efficient mechanisms;
- preventing various types of DoS attacks;
- a means of verifying the origin and integrity of routing information data with low computation overhead;
- a terminal should trust no one other than itself, so as to avoid blackmailing;
- delivering services in a more secure, more efficient and more general manner relative to previous work in the field, such as not relying on a trusted hardware and requiring low processing power.

2.3.3. Route-Request in ARIADNE

A route-request (RREQ) in ARIADNE is in the form:

(RREQ, S, D, id, hash-chain, node-list, MAC-list),

and at the time a RREQ packet is first created these fields have the values:

- S is the address of the source/initiator;
- D is the address of the destination/target;
- *id* is a unique identifier for the current route-discovery;
- *hash-chain* is initialized to $MAC_{K_{SD}}(RREQ, S, D, id)$, a message authentication code computed over the tuple $(RREQ, S, D, id)$ using K_{SD}, the secret-key between the initiator and the target;
- *node-list* and *MAC-list* are empty lists.

Propagating a RREQ: Upon receiving a previously not seen RREQ packet, a terminal, say A, if it is not the target for that RREQ, appends itself to the *hash-chain* value and

hashes the result; thereby computing the new *hash-chain* value:

$$h_1 = H[A, h_0].$$

Then, it appends itself to the *node-list* and also computes the new MAC value as:

$$M_A = MAC_{K_{AD}}(RREQ, S, D, id, h_1, (A), ()).$$

Now A appends this new MAC value to the end of the *MAC-list*, and broadcasts the RREQ as:

$$(RREQ, S, D, id, \underline{h_1}, (\underline{A}), (\underline{M_A})).$$

Note that we mark the changed fields by underlining them.

Each intermediate node acts the same: computes a new *hash-chain* value by firstly appending itself to the old one; appends itself to the *node-list*; computes the new MAC value over this new version of the RREQ, and appends this new MAC value to the end of the *MAC-list*; and then rebroadcasts the resulting packet.

Validating a RREQ: Upon receiving a RREQ packet, a terminal, say D, if it is the target for that RREQ, validates the *hash-chain* using K_{SD}. If this check turns out to be valid, D subsequently validates the *MAC-list* using its pairwise secret-keys for each intermediate node. If *MAC-list* is also validated then a route-reply (RREP) is created. If not valid, the packet is simply dropped.

See Figure 2.3 to examine RREQ mechanism in ARIADNE schematically.

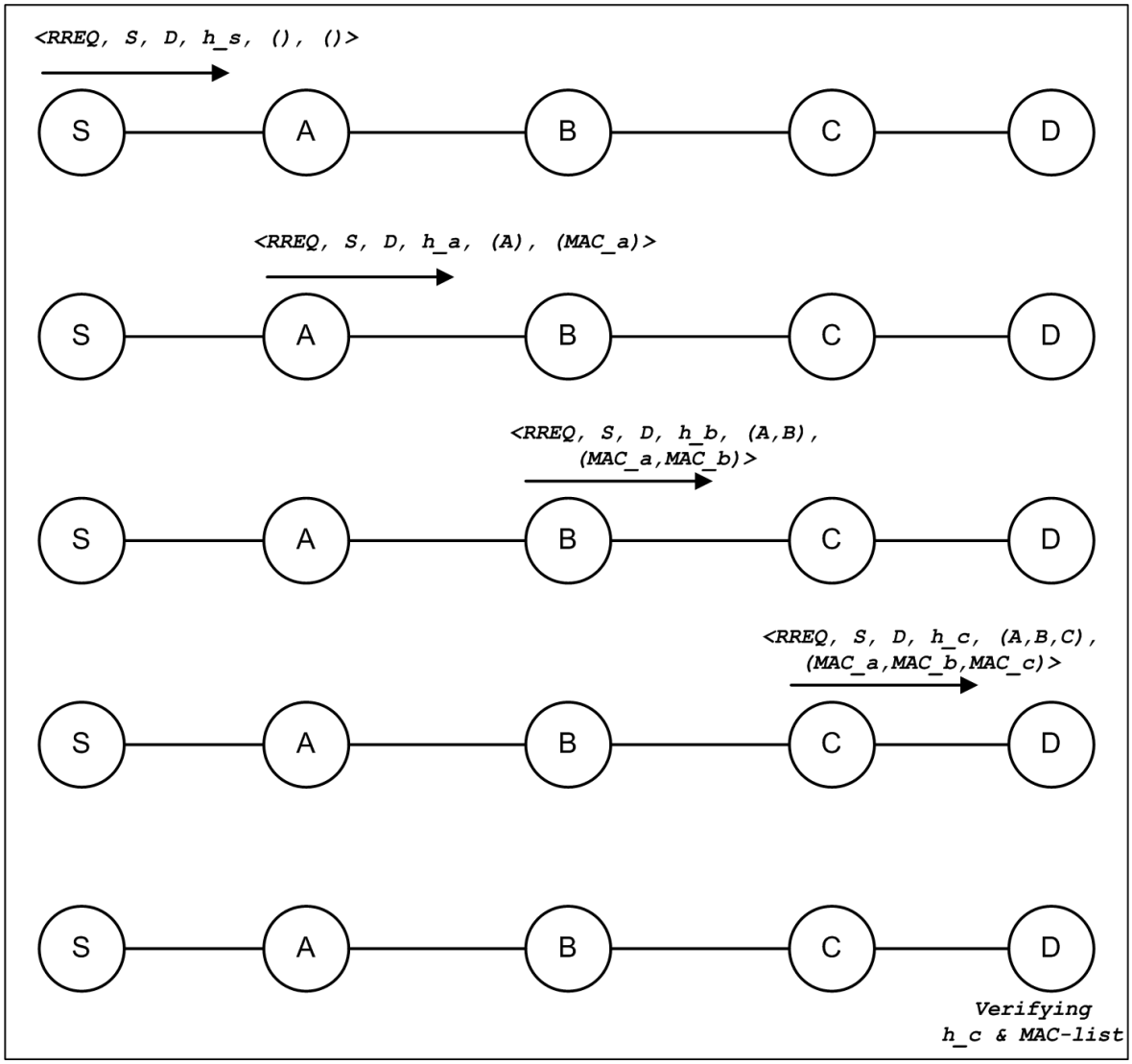

Figure 2.3. Normal operation in ARIADNE; propagating the RREQ.

2.3.4. Route-Reply in ARIADNE

The destination terminal D, upon receiving a valid RREQ of the form:

$$(RREQ, S, D, id, h_3, (A, B, C), (M_A, M_B, M_C)),$$

creates a RREP in the form:

$$(RREP, D, S, (A, B, C), M_D),$$

where M_D is calculated by:

$$M_D = MAC_{K_{DS}}(RREP, D, S, (A, B, C)).$$

Then the destination unicast this RREP by following the *node-list* in the RREQ reversely.

Forwarding a RREP: An intermediate terminal upon receiving a RREP packet, if it is not the target for that RREP, simply forwards the packet to the next hop by following the *node-list* in the RREP reversely.

Validating a RREP: Upon receiving a RREP packet, a terminal, say S, if it is the target for that RREP, validates the *MAC-list* field using K_{DS}: the secret-key between the target and the initiator. If validation is ok, the route is received; if not, the packet is simply dropped.

See Figure 2.4 to examine RREP mechanism in ARIADNE schematically.

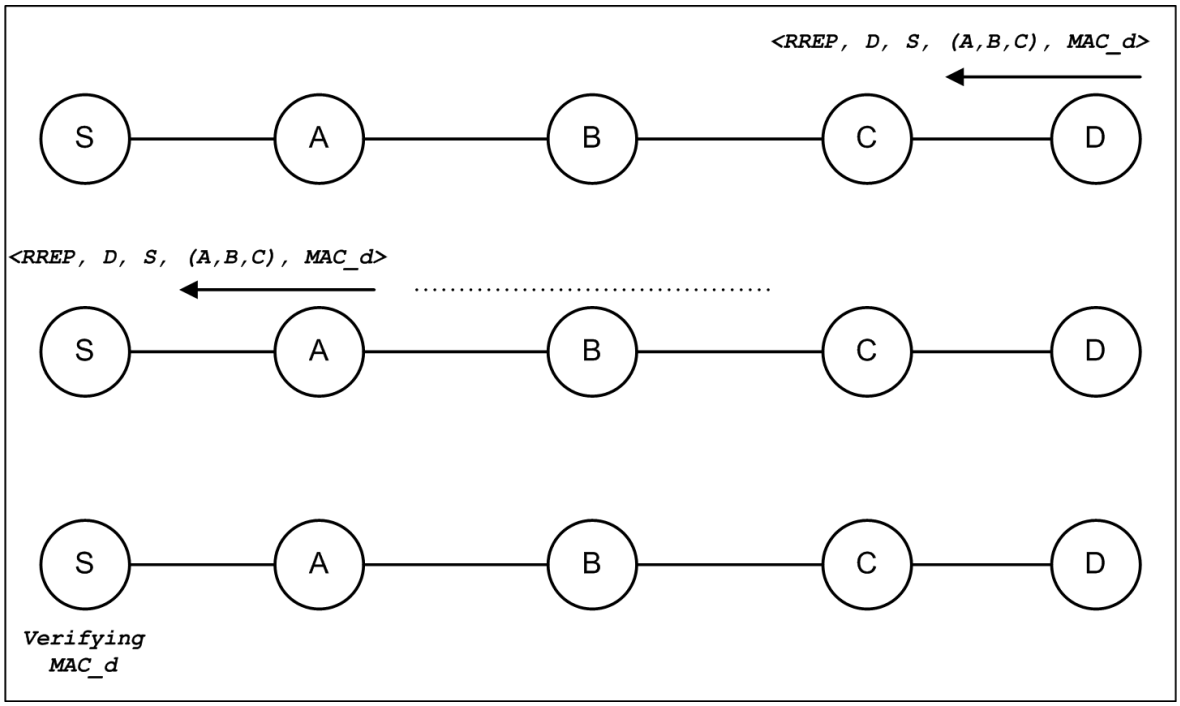

Figure 2.4. Normal operation in ARIADNE; forwarding the RREP.

2.3.5. Route-Error in ARIADNE

A route-error (RERR) packet in ARIADNE is of the form:

(RERR, sending-address, receiving-address, error MAC),

where the *sending-address* is the address of the intermediate terminal which encountered the error, *receiving-address* is the address of the intended next hop, and the *error MAC* is the MAC computed over the preceding fields of the RERR, using the secret-key between the sender of the RERR and the source address of the original packet.

Forwarding a RERR: An intermediate terminal upon receiving a RERR packet, if it is not the target for that RERR, simply forwards the packet to the next hop by following the *node-list* in the original packet reversely.

Validating a RERR: A terminal, say S, upon receiving a RERR packet sent by a terminal, say B; if it is the target for that RERR, validates the *error MAC* field using K_{SB}. If validation is ok, all the routes using the indicated link are removed from the route cache of S; if not, the packet is simply dropped.

2.3.6. Security Discussion

ARIADNE uses *hash-chains* and *MAC-lists* to provide security. A hash-chain is a list of hash values, linked together cryptographically. It is created by taking an initial seed and incrementally hashing it n times; where n is said to be the length of the chain. The usability of this hash-chain comes from the fact that given any element from the hash-chain, it is infeasible to calculate any previous elements from the chain. The target can check the authenticity of any RREQ through the use of hash-chains. The initiator initializes the first element of the hash-chain to a MAC computed over $(RREQ, S, D, id)$, using K_{SD}. Any intermediate node takes the current hash-chain value, append itself, and re-hashes before rebroadcasting the packet. In this way, the target can make sure that the node-list in the RREQ is valid; by reconstructing the

MAC computed over $(RREQ, S, D, id)$ using K_{SD}, and then reconstructing the hash-chain from scratch by using the node-list, and finally checking whether it is equal to the hash-chain in the packet. The security here relies on the fact that no node can hear a RREQ without itself listed[1].

A MAC-list is a list of MAC values that are stored separately. MAC values are used for both authentication and integrity purposes. Each terminal computing a MAC value in ARIADNE, does so over the previous fields of the packet by using the shared-key between itself and the target node. During a RREQ propagation, these MAC values are appended to the existing MAC-list; thereby increasing the packet-length while traversing the route. In this way, the target can make sure that the packet is not modified along the route; by recalculating MACs for each hop over the route and comparing each calculated value with the value in the RREQ packet. Furthermore, if someone over the route modifies the RREQ, the target knows which node it is[2]. Also, the initiator of the route-discovery can make sure that the RREP received is valid; by recalculating the MAC over the RREP using K_{DS}, and then comparing it with the value in the received RREP packet.

By these two mechanisms described, ARIADNE provides authentication of the initiator and the target of the route-discovery; and makes sure that any modifications on a route-discovery packet will be detected at the target or at the initiator; and furthermore, no node over the route can remove previous nodes from the node-list in a RREQ.

[1]However, Buttyán demonstrated a way of making the attacker "hear" such a RREQ, by transmitting a previous hash-chain value hidden in the RREQ packet [1]. Our approach also uses Buttyán's approach, but with two compromised nodes instead of one; so that a repeating identifier in the node-list can be avoided.

[2]This information can later be used for detection and reaction purposes.

3. MODEL CHECKING ARIADNE USING SPIN

3.1. Main Objective

Our main objective in this thesis is to apply model checking approach on the secure routing protocol ARIADNE for ad hoc networks, in order to flag a sequence of possible events in the protocol leading to a new attack of type *Active-2-2*. The authors in [52] state that by using secure hash chains and message authentication codes, ARIADNE ensures the fact that no node can remove other nodes from the *node-list* of a RREQ packet without being noticed by the destination. This statement is our security property of interest. So, our property to check is that; *no intermediate node in a route-discovery process is able to remove nodes from the RREQ.*

In fact, Buttyán *et al.* described an *Active-1-2* attack on ARIADNE such that if there is a compromised node and an attacker node over the route to the destination, these nodes are able to remove the nodes between them from the RREQ [1]. By a similar approach, but with a network configuration consisting of two compromised nodes instead of one; our aim is to use SPIN to find a sequence of possible events in ARIADNE leading to a new attack, which is of type *Active-2-2*, where the two compromised nodes collaborate to remove all the intermediate nodes from the RREQ. This is realized through modeling a legitimate node in ARIADNE, and two distinctly behaving compromised nodes with own legitimate keys; so that SPIN will find a counter-example in the protocol which shows how the claim, "if a node receives a valid RREP then the route in the RREP does really exist", can be violated in a five-node linear network topology. Buttyán *et al.* also stated that if each node checks the *node-list* for a repeating identifier then their attack can be prevented. But the attack we describe here can not be prevented in this manner.

3.2. On Effective Model Checking Using SPIN

Describing a formal specification of any system for model checking purposes requires an abstraction process as the most important part of the work, since the most compelling drawback of model checking is the computational complexity; and a model checking tool, e.g., SPIN, already has very efficient inner mechanisms to combat this problem (For details, see Section 2.2.3). State-of-the-art model-checkers only require that the user gives a specification of the model, so the rest is only a push-button action. All remaining to the user is making abstractions of the system in such a way that the system eludes unnecessary details, but still captures the essence of the solution, and no more. What we are looking for is the smallest sufficient model sufficient to verify the properties that we are interested in. Holzmann discusses that [95], for effective model checking one should first decide which aspects of the design are important and require verification. In this light, in the formal specification phase, we have not considered some of the main aspects of ARIADNE; such as the route-maintenance mechanism, since it is irrelevant to the flaw we are trying to show. In addition, T. Ruys discusses some SPIN optimizations one should consider while building a verification model [131]. He makes very helpful comments and recommendations to reduce the complexity of the model under construction. We have tried to stay strictly attached to his advices throughout the modeling process.

3.3. Assumptions for Our Model

We inherit all the assumptions of the protocol (See Section 2.3.2). As explained in Section 3.2, we have not considered the route-maintenance mechanism to strengthen the focus on our issue of interest, namely, the route-discovery phase of the protocol.

We have not considered mobility in our model since we aim at showing an execution of events in the protocol leading to a possible attack in a special configuration, namely, when there is one sender and one receiver with the two of the three intermediate nodes compromised and furthermore, there is one legitimate node between the two compromised terminals. This special configuration leads to a linear topology

in the most simple case. There are more complicated network topologies where it is still possible to have this configuration inside. We will discuss this issue in Section 4. But for the sake of simplicity and verification performance, we assume a linear topology.

We have not considered timers either, since the authentication mode of the protocol is through MAC functions. If we had considered ARIADNE with TESLA, we would have probably needed timers in the model.

The ad hoc network protocol verification approaches so far generally use at most five nodes in the model, since five nodes may represent such different network configurations that a verification done provides a great probability of good behavior for larger networks[1][131]. We also use five nodes in our model.

We assume that any terminal may non-deterministically initiate a route-discovery for a non-deterministically chosen destination. The RREQ packets sent may also be non-deterministically timed-out.

3.4. Modeling Approach for the Ad Hoc Network

While specifying a Promela model to verify a system, the general approach is to deal with two distinct modelings [119]:

- the modeling of the protocol rules and functions, and
- the modeling of the intruder behavior.

The modeling of the protocol rules and functions involves the network primitives such as communication channels, route-discovery messages, and cryptographic structures; along with legitimate ARIADNE participants. Modeling approaches for route-discovery and cryptographic structures are often performed in the literature; and we will discuss the alternative modelings and our preference in Section 3.4.2, Section 3.4.3, and Section

[1]However, this is not an issue for our approach since we target demonstrating an attack on the protocol for a special network configuration, not verifying the protocol for all possible configurations.

3.4.4. However, modeling of the second part constitutes more challenge; which we will discuss in Section 3.4.5 and Section 3.4.6. First of all, we build some necessary constructions for the model.

3.4.1. Necessary Constructions

Our model consists of five nodes, which can be defined in PROMELA as:

```
#define N    5
```

Each node will have a table keeping $(initiator, id)$ pairs of the recently broadcast RREQs:

```
#define MaxTableSize    30

typedef RouteReqTableType {
        byte id[MaxTableSize];
}
```

The number of slots in the terminals' tables keeping track of $(initiator, id)$ pairs of the recently broadcast RREQs is assumed to be 30. If this buffer gets full then the node begins writing from the first slot again. This is safe since it is not much probable that a previously seen RREQ will be seen again after 30 other unseen RREQs. Note that only id is enough for keeping track of the $(initiator, id)$ pairs since we generate id's globally, i.e., any id identifying the current route-discovery appears only once globally.

The terminals in the network also have some common characteristics. They have unique identifiers, tables to remember $(initiator, id)$ pairs of the recently broadcast RREQs, their own private-keys for each other node in the network; and they know who their neighbors are. We describe this information in PROMELA as:

```
typedef NodeDataType {
        byte myID;
        RouteReqTableType RecentRouteReqTable;
        byte myNeighbors[N];
        int myPrivateKeys[N]
}
```

There are totally five nodes in the network so the variable *myID* of type *byte* is sufficient for identifying the terminals. Each node has the knowledge of its neighbors stored in the byte array *myNeighbors*, for purposes of modeling the broadcast system through the native unicast communication system in PROMELA[2]. Since we will not be interested in real cryptographic functions, *myPrivateKeys* field is a symbolic field which will never be used.

3.4.2. Modeling Cryptographic Structures

The concept of protocol verification targets possible security flaws in the protocols, not flaws in the cryptosystems used by the protocols. Therefore, cryptography is typically modeled in an abstracted form with the assumption that it works perfectly as intended, meaning [119]:

- an encrypted message can only be decrypted with the corresponding key;
- by examining the encrypted message, one can not have an idea of what the key can be;
- there is sufficient redundancy in messages so that the decryption algorithm can detect whether a ciphertext was encrypted with the expected key.

Although these are generally not true for real cryptosystems, they are useful in the sense that the flaws of the protocol itself can be isolated well, and also the verification gets simpler by eluding the burden of identifying cryptographic flaws.

[2]This communication issue will be discussed in Section 3.4.4.

As we have discussed before, the main cryptographic structures used in ARIADNE are *hash-chains* and *MAC-lists*. Since each new hash-chain value is constructed by each node appending itself to the end of the previous value and re-hashing, it is convenient to represent this hash-chain as a simple list in our model. So, we can model the process of computing a new hash-chain value as simply adding the node's identifier to the existing list. This hash-chain can have at most the length equal to one less than the number of terminals in the network, i.e., $N - 1$.

```
typedef HashListType {
        byte chain[N]
}
```

A MAC-list is expanded by adding the new MAC value to the end the previous list in a per-hop basis. So, we can also represent this structure as a simple list:

```
typedef MacListType {
        byte secret[N];
        byte secretHopCount;
        byte chain[N]
}
```

Here, the variable *chain* is the list we have mentioned; and the other fields are used for the purposes of modeling the approach in Buttyán's attack. We will use these fields to hide a hash-chain value in the corresponding MAC field of a MAC-list[3].

3.4.3. Modeling Messages in the Network

Since our issue of interest is the route-discovery process, our model has only messages of type RREQ and RREP:

```
mtype = { RREQ, RREP }
```

[3]Buttyán's attack is explained in detail in Section 3.4.6.

Typically, there are two different approaches while modeling multiple types of messages. The first is that each packet type is modeled distinctly, so the communication channel through which these messages will pass should also be modeled distinctly. The second approach involves redundancy but eases the modeling; since both types of messages are modeled using the same structure and therefore we need no more than one communication channel for both types. We follow the second approach:

```
typedef RoutingPacketType {
        mtype flag;
        byte sourceID;
        byte destID;
        byte id;
        HashListType hashList;
        byte nodeList[N];
        MacListType macList;
        byte hopCount;
        byte macRoute[N]
}
```

Here, *flag* denotes the type of the packet; *id* is a globally unique route-discovery identifier of the packet; *hopCount* is the number of hops the packet is forwarded; and *macRoute* is for storing the route on which the MAC of the RREP is computed.

3.4.4. Modeling Communication Between Nodes

Communication in PROMELA is provided by *message channels*. These channels have a unicast nature connecting two communicating nodes, and have the ability of buffering a number of M messages. We define M to be 1:

```
#define M    1
```

It is necessary to choose this number small enough to let the communication pro-

ceed, but not larger than that; since choosing it large directly increases the verification complexity.

For our five-node network, we need five communication channels so that each node has a one-way channel to communicate with the others:

chan channel[N] = [M] of { *RoutingPacketType* }

Here, we have an array of channels whose i'th element belongs to the node with id equal to $i+1$ in the network. However, in order to model the neighbor-sensing phase of the terminals, one more channel is needed:

chan topologyChannel[N] = [0] of { *NodeDataType* }

This channel tells each terminal who its neighbors are, at the beginning of the protocol run. So the broadcasting function can be realized by the terminals through simulating it by unicasting to each of the neighbors throughout the protocol run. Note that this channel is only used once and is never needed again. If we had considered mobility, it could have been possible that we distribute new topology information using this channel each time a topology change occurs.

3.4.5. Modeling Legitimate Nodes

Each node has its characteristic node-data and immediately sets his network identifier to the value that *init* process sends to it in variable _ID:

NodeDataType myData;
$myData.myID = \_ID;$

It should be noted that *init* is the first process to run in a SPIN model and it creates all the other processes by optionally sending them parameters. After initializing the processes, *init* sets the topology (See Figure 3.1) and the terminals learn this

topology information through the function *learnNeighbors*.

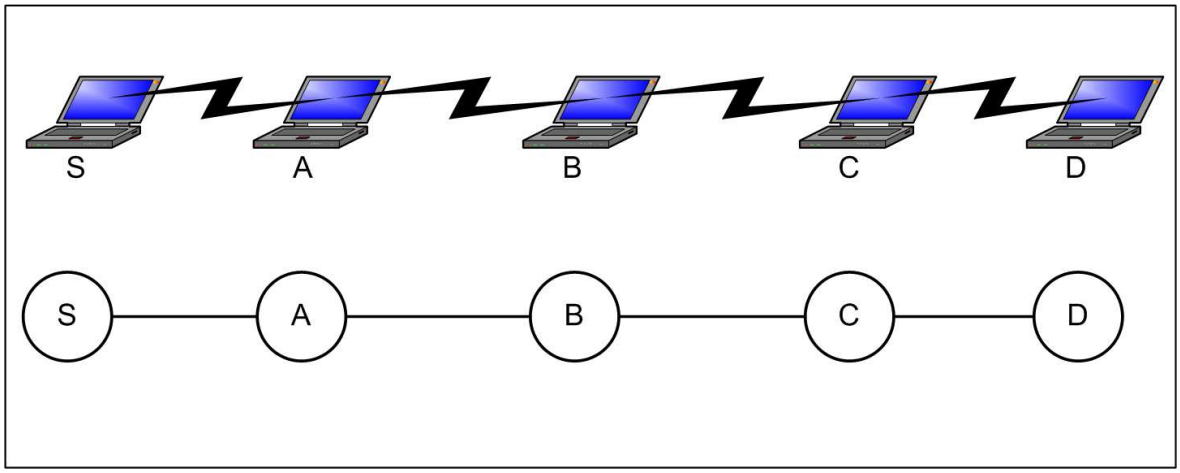

Figure 3.1. Topology assumed.

In the first place, a node with node id equal to $i + 1$ gets topology information using the *topologyChannel[i]*. This happens for all i; and only then the processes may begin executing the protocol.

In our model, there is a main loop, which the terminals execute *ad infinitum*, and is defined with the label *MAIN_L*. A legitimate node first checks his channel for any received packets. If there are no received packets in the channel, the terminal randomly assigns a destination for initiating a route-discovery. But if the terminal has already initiated a route-discovery for that destination before, it checks whether a timeout for that RREQ occured. If he has not initiated a route-discovery for that destination before, or the previously initiated RREQ is timed out then he generates a RREQ for that destination and broadcasts. If there is a received packet in its channel, the terminal takes a course of actions according to the type of the message. If it is a RREQ then it first checks whether the packet has been seen before. If seen, it is simply dropped. If not, the terminal checks whether the packet is destined to himself. If not, it is rebroadcast after the modifications of *hashList*, *nodeList*, *macList* and *hopCount* fields. If the terminal is the destination for that RREQ then it checks validity of the *hashList* and the *macList* fields. If valid, the terminal creates a RREP and forwards; if not, the packet is simply dropped. If the received packet is a RREP then the terminal checks whether the packet is destined to himself. If not, it is forwarded to the next hop with no modifications. If so, the validity check of the *macList* field along with

the *macRoute* field is performed. If valid, the route in the RREP, i.e., *nodeList* field is accepted as a valid route. If not valid, the packet is simply dropped. These operations can be examined schematically in Figure 3.2.

Note that in the real execution of the protocol, RREP for any RREQ may not return in time so a RREQ-timeout may occur. In our model, this concept is not modeled using real timers; instead, a RREQ-timeout may take place non-deterministically; which is enough to capture the essence of operation.

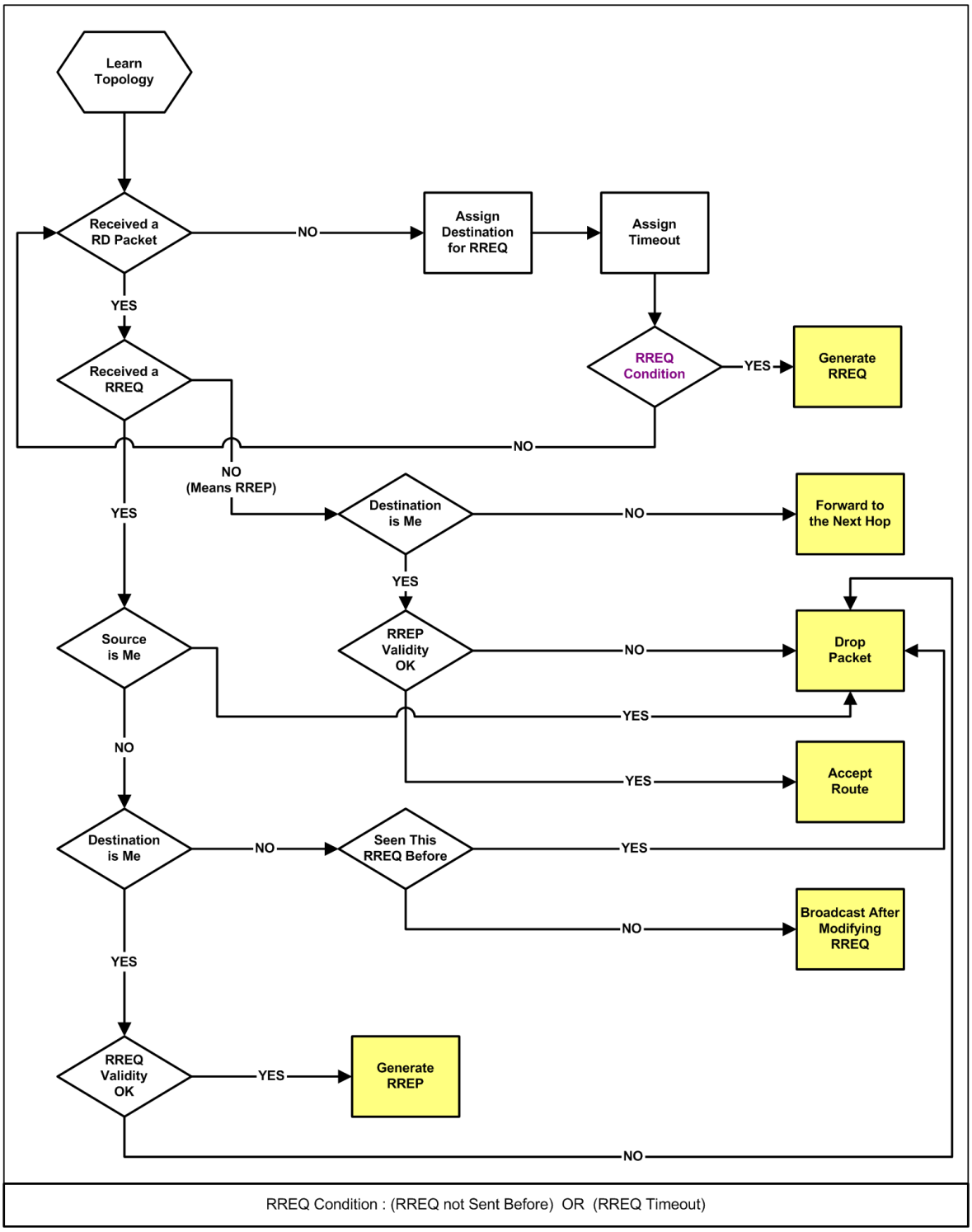

Figure 3.2. Operations of a legitimate terminal in our model.

3.4.6. Modeling the Compromised Nodes

The Dolev-Yao model [132] represents an attacker that can eavesdrop, intercept, and inject any message into the system and is only limited by the constraints of the cryptographic methods used in the system. The intruder has complete control over the network and he is able to compose new messages using his initial knowledge and the messages collected from the network traffic during the protocol run. The modeling of a pure Dolev-Yao intruder for model checking purposes is hard to realize and most threat models in the literature simplify it; since a Dolev-Yao intruder may behave in a highly non-deterministic manner, resulting in a very complex verification model. Paolo Maggi and Riccardo Sisto [119] offer complexity reduction techniques based on a preliminary data-flow analysis to build a simplified model for the attackers. By this method, one can achieve an intruder model which does not flood the network with nonsense messages; instead, performs potentially beneficial actions. The idea of a simplified intruder model may exclude some possible attacks from the model for most of the cases, but otherwise it is very hard to deal with the resulting complexity.

A popular research approach to model an intruder involves the fact that principals in the network do not communicate with each other directly but all the messages they send are intercepted by the attacker which eventually will forward them to the right addresses [119]. We do not follow that approach; instead, in our model the compromised nodes do not hear any other information than they are naturally supposed to hear. Furthermore, they act almost the same as a legitimate node. What they do more is that the first compromised node tries to hide the *hash-chain* value in the place where it should have normally appended a MAC-value. This concept is realized in our model by assigning the value of the *hashList* field to the *macList.secret* field and placing a special mark in the proper *macList.chain* element, into which we should have normally placed our MAC-value. By this way, some other adversary using the same identity which is over the route and is aware of this possible attack will understand that a hidden hash-chain element was sent when it saw the special mark in the MAC-value field; thereby being able to remove the nodes between the two adversaries using this hidden hash-chain value. The target will then verify the RREQ since there is nothing wrong in

it and then in the second phase of the attack the second adversary will add the nodes it has removed to the *node-list* again, in order to provide a harmless forwarding process towards the initiator, using this added source route. Then the first adversary removes those added nodes from the *node-list* and renders the packet to the state on which the MAC was calculated. The result is that the initiator accepts this route as if it is real, where actually it is not. The RREQ propagation phase and RREP forwarding phase of this attack can be examined in Figure 3.3 and Figure 3.4, respectively.

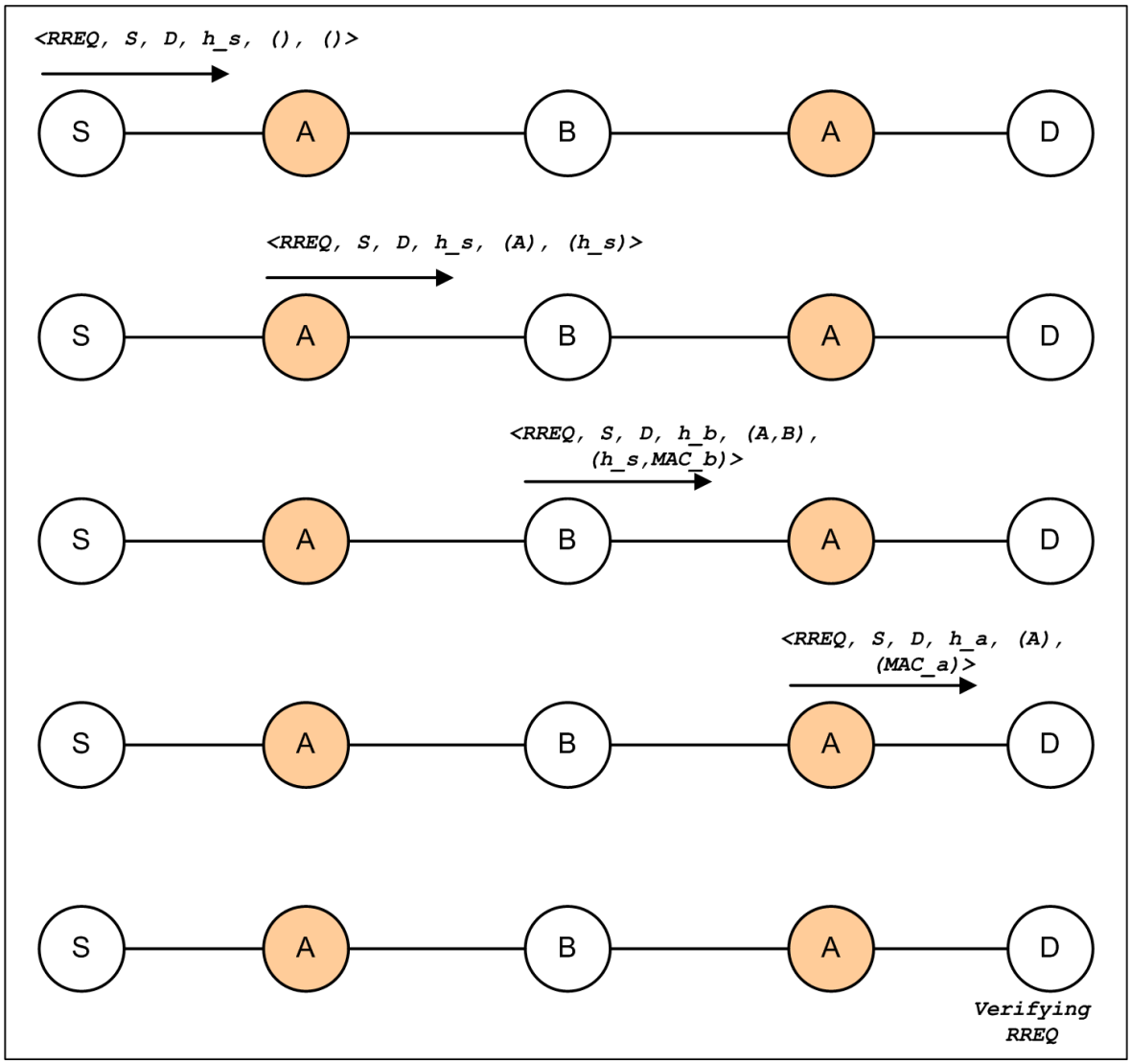

Figure 3.3. Buttyán's *Active-1-2* attack on ARIADNE; propagating the RREQ.

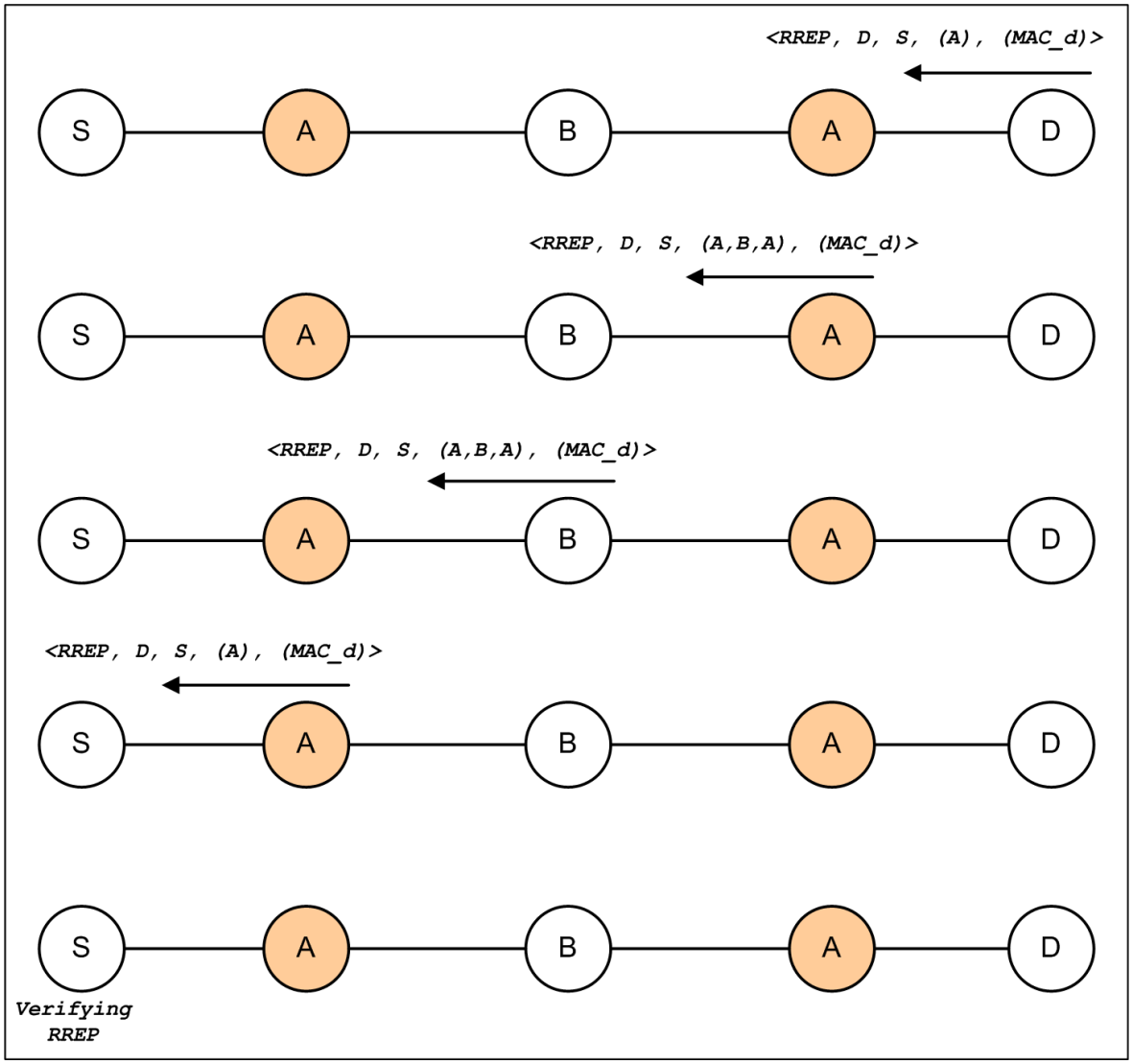

Figure 3.4. Buttyán's *Active-1-2* attack on ARIADNE; forwarding the RREP.

This attack was first described by Buttán *et al.*, and they proposed that the terminals in the protocol can prevent this attack if they check for a repeating identifier in the RREP forwarding phase. What we intend to add is that if there are two compromised nodes instead of one, this attack is still possible to perform and furthermore it is not possible to prevent this attack by checking for a repeating identifier, since there is none. What we have done is to model the possible behaviors of this approach using a modeling and verification tool, such that the tool itself will show us an execution sequence in the protocol which leads to the realization of this attack.

3.5. Simulation Phase

As we have discussed before, SPIN's simulation mode can be used to get an impression of the model and shape the further modeling. For simulation purposes, one can use XSPIN graphical interface which operates independently from SPIN itself (See Figure 3.5).

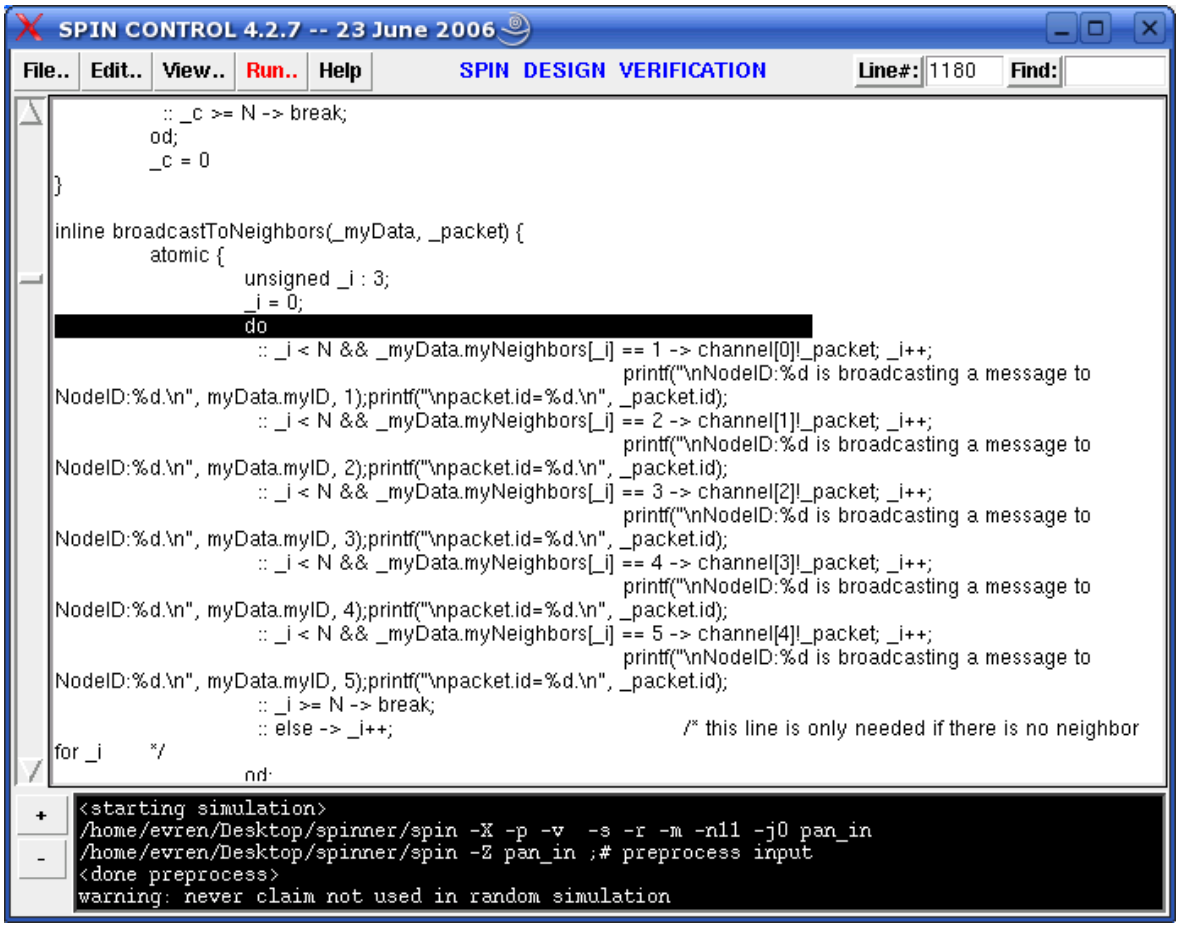

Figure 3.5. XSPIN graphical interface.

This tool can greatly ease the modeling and verification phases by providing graphical displays of message flows, time sequence diagrams, and a parse tree of the model. A *message sequence chart (MSC)* shows the simulation steps with each line representing a process in the model. When a process sends a message to another process, a new box appears on the process line with an arrow pointing to the process which receives that message. Each box has a state number which is uniquely assigned by SPIN, and SPIN also writes the message contents on each arrow. The first line in an *MSC* always represents the process *init*, and the other lines are lined up in order

of unique process identifiers given to them by SPIN according to their creation times. A part of the *MSC* of our model taken from a simulation run can be examined in Figure 3.6. In the figure, it can be observed that *init* process is sending the topology information to each of the five node in the network before any further message traffic begins. Only then, the processes begin executing the protocol by sending RREQs non-deterministically and randomly to each other. Here it should be noted that the concept of randomness is granted by non-determinism in SPIN. The process of choosing a random destination involves non-deterministically sending a message to one of the other processes. It can be observed that the first process initiates a route-discovery at state-number 305, and the third process initiates another route-discovery at state-number 410. The second process performs the required operations on the RREQ packet it has heard from the first process and then rebroadcasts. Simulation run goes on like this. At any time the user can stop the simulation and check which part of the PROMELA code is being executed at that instance.

XSPIN has three types of simulation styles:

- *Random simulation* can be performed where every non-deterministic action in the model is taken in a random manner.
- *Guided simulation* can be performed where the simulation steps are read from a trail file that SPIN has generated upon finding any violation of a property in the model. This type of simulation shows a complete sequence of events leading to the violation condition found.
- *Interactive simulation* can be performed where the user is asked to determine every non-determinism throughout the simulation run.

The purpose of random simulation is that the user can observe different examples of asynchronous execution sequences of the protocol in a graphical environment and can notice any anomaly in the behavior of the processes in an easy and quick manner. Simulation phase can be rerun each time with a different seed of randomness until the user is persuaded that the modeling of the behavior is successful. We have done several random simulations for our model before defining our security property and trying to

verify it. After the verification run when SPIN has found a violation to our claim, we have used the guided simulation mode to examine the events leading to the violation condition.

3.6. Specifying the Security Property for Verification

Our issue of interest to check the protocol against is that if it can really prevent the intermediate nodes from removing other terminals from the node-list, in an *Active-2-2* adversary environment. But we should rephrase this property so that SPIN will be able to understand what it means.

At the end of a route-discovery process, if the returned route is an inexistent route, then that means either an intermediate node has removed some nodes from the node-list, or an intermediate node has added some nodes into the node-list, or someone has done both[4]. But no possible behavior of adding nodes into a node-list is present in our model. Hence it must be true for our model that if the returned route is an inexistent route, then this means an intermediate node has removed some nodes from the node-list.

By this discussion, now we can be sure that *if we find an inexisting route returned from a route-discovery, then an intermediate node has removed some nodes from the node-list.* So, the property that "*if some route is accepted by a legitimate node, then that route does really exist*" will be our security claim to be verified.

[4]Note that we exclude the possibility that the target of the route-discovery generates a false RREP since the protocol trusts communicating node pairs.

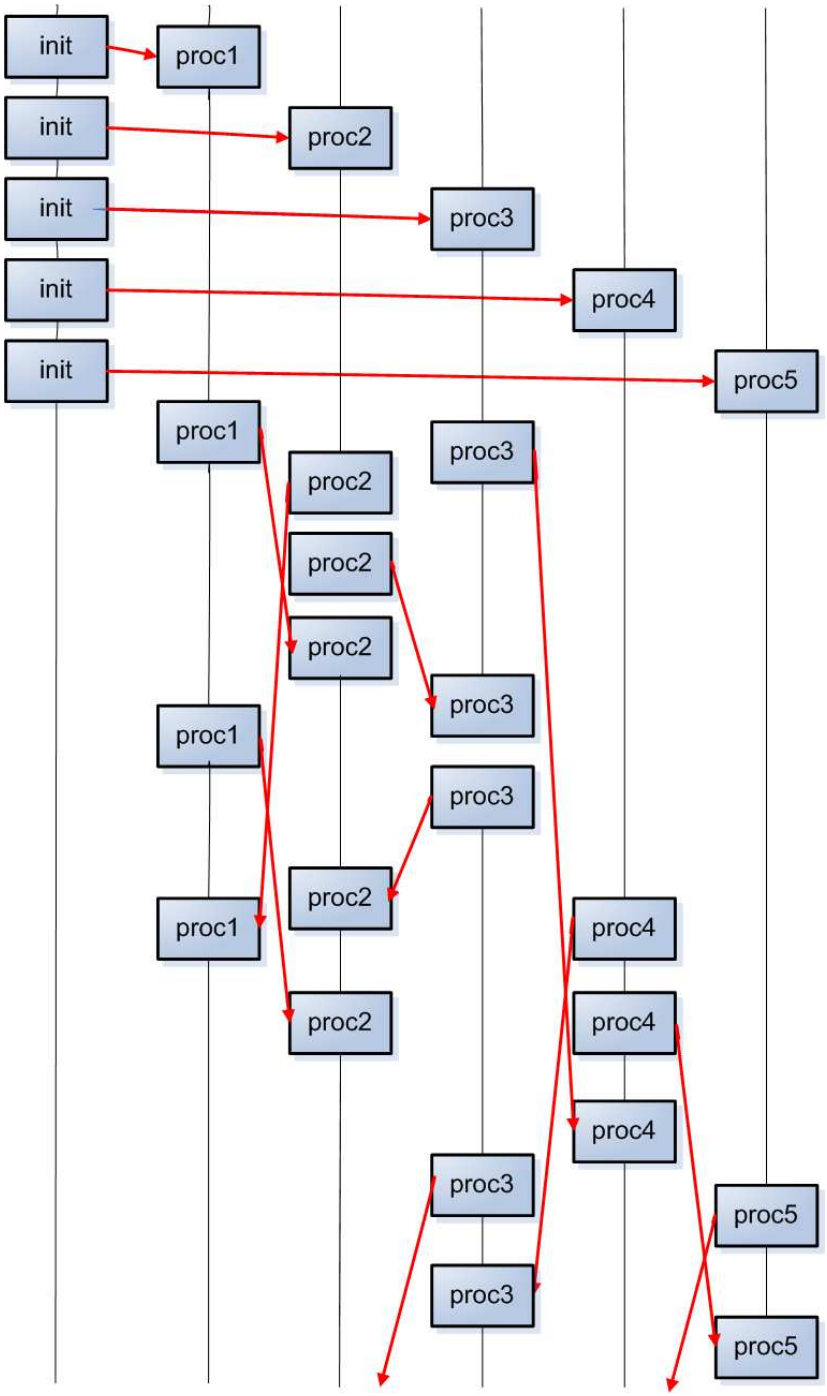

Figure 3.6. Part of a random simulation for our model performed in SPIN's simulation mode.

In our topology, this property is logically equivalent to the statement:

"*(If node 1 accepts a route from node 5 then the route must be (2,3,4) OR*
if node 1 accepts a route from node 4 then the route must be (2,3) OR
if node 1 accepts a route from node 3 then the route must be (2))
AND
(if node 3 accepts a route from node 5 then the route must be (4))
AND
(if node 5 accepts a route from node 1 then the route must be (4,3,2) OR
if node 5 accepts a route from node 2 then the route must be (4,3) OR
if node 5 accepts a route from node 3 then the route must be (4))".

We can either manually code this property in PROMELA, or automatically generate from LTL formulas using SPIN's built-in translator.

3.7. Verification Phase

Once having built the formal model and specified the property to check, the next thing is to compile a verifier for our model by using SPIN. There are a few compile-time directives but they can properly modify the default behavior of the verifier to achieve specific effects. Some of which we have used while searching the best one to suit our needs are:

- *-DBFS*: This is used for a breadth-first search but uses more memory and lets only the safety properties be verified. But it is the easiest way to find a short error path.
- *-DMEMLIM=N*: Lets the verifier use memory bounded by N Megabytes.
- *-DNOBOUNDCHECK*: Disables the check on array indices to improve performance.
- *-DSAFETY*: No cycle detection is targeted, thereby improving performance.
- *-DBITSTATE*: Makes use of the *bitstate storage* algorithm instead of exhaustive storage. This option greatly reduces the run-time of the verifier, but only has a

partial coverage of the state space. The coverage quality is expressed in the form of a hash-factor after the verification run.

- *-DCOLLAPSE*: By state descriptor compression, this method increases run-time but significantly reduces the memory requirements.
- *-DMA=N*: Uses the *minimized automaton storage* method for the state descriptors. Leads to a very significant reduction in memory requirements at the cost of a very significant increase in the run-time requirements.
- *-DVECTORSZ=N*: Tells the verifier that size of the state descriptor is N bytes at maximum.
- *-DREACH*: Changes the search algorithm to make sure that there is no safety errors within the run-time depth limit set by the run-time directive *-m*.

After the compilation of the verifier, one can make use of some proper run-time directives of SPIN such as:

- *-i*: Enables an iterative search method to look for the shortest path to an error. For identifying the shortest possible error path, the compile-time directive *-DREACH* must be used. For safety properties, this option is guaranteed to work.
- *-mN*: For a depth-first search verification, this option sets the maximum search depth to N.
- *-n*: Suppresses the listing of unreached states at the end of a verification run.
- *-wN*: The default size of the hash table can be changed to 2^N slots, up to a maximum value of *-w32*.

3.7.1. Dealing with Complexity

SPIN can generate automata for the modeled processes. So, by examining these automata one can have an idea of how large the state space is. In Appendix A, Figure A.1, Figure A.2, and Figure A.3 show the automaton of a legitimate node which has more than 650 states, the automaton of the first compromised node which has more than 850 states, and the automaton of the second compromised node which has more

than 950 states, respectively. This analysis tells us that our state space is around $650^3 * 850 * 950 = 221,759,687,500,000 \simeq 2,22 \times 10^{14}$ states.

Since our model has a very large size, we have decided to try some directives to help reduce the memory requirements. We have tried to verify our model using vector size of 2048 bytes, memory limit of 1.5 GB and the options 'COLLAPSE', 'NOBOUNDCHECK', 'SAFETY'. The corresponding compilation command is as follows:

```
gcc -o pan  -DVECTORSZ=2048 -DCOLLAPSE -DNOBOUNDCHECK
            -DMEMLIM=1500 -DSAFETY pan.c
./pan -n
```

Here, the run-time directive '*n*' is used to supress the listing of unreached states at the end of the verification run. This trial has led to an *out of memory* error after visiting 44 million states. This experience showed us that we really have a huge state space. After that, we have continued trying memory reduction techniques, again by using the same vector size, the same memory limit and the options 'DMA', 'NOBOUNDCHECK', 'SAFETY'. The corresponding compilation command is:

```
gcc -o pan  -DVECTORSZ=2048 -DMA=4756 -DNOBOUNDCHECK
            -DMEMLIM=1500 -DSAFETY pan.c
./pan -m15000 -n
```

The additional run-time directive '*m15000*' is used to conduct a depth-first search verification with the maximum search depth 15000. We have also tried the compile-time options 'DREACH', 'NOBOUNDCHECK', 'SAFETY' with the run-time option '*i*' which initiates an iterative search method to look for the shortest path to an error:

```
gcc -o pan  -DVECTORSZ=2048 -DREACH -DNOBOUNDCHECK
            -DMEMLIM=1500 \-DSAFETY pan.c
./pan -n -i
```

However, none of them could give us a result as good as 44 million states. Only then we have tried our best weapon at the expense of a partial coverage of the state space: the bitstate storage algorithm. It should be noted that we do not aim a verification but an error trail of a possible violation; so the trouble of partially covering the state space is not an issue for us as far as we can find an error trail. For using bitstate hashing with a 2^{31} slotted hash table, we have used the following commands:

```
gcc -o pan  -DVECTORSZ=2048 -DBITSTATE -DNOBOUNDCHECK
            -DMEMLIM=1500 -DSAFETY pan.c
./pan -n -w31
```

The result is that the verifier has visited 673 million states and flagged a violation of the claim at depth 2212. The output of SPIN's verifier is:

```
pan: claim violated! (at depth 2212)
pan: wrote evren_ariadne_30_06_06_M=1.pml.trail
(Spin Version 4.2.6 -- 27 October 2005)
Warning: Search not completed
     + Partial Order Reduction

Bit statespace search for:
     never claim             +
     assertion violations    + (if within scope of claim)
     cycle checks            - (disabled by -DSAFETY)
     invalid end states      - (disabled by never claim)

State-vector 1816 byte, depth reached 9999, errors: 1
```

```
6.73038e+08 states, stored
2.63112e+08 states, matched
9.3615e+08 transitions (= stored+matched)
1.66947e+09 atomic steps

hash factor: 3.19073 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
1224928.912      equivalent memory usage
                 for states (stored*(State-vector + overhead))
536.871 memory used for hash array (-w31)
0.360   memory used for DFS stack (-m10000)
0.803   other (proc and chan stacks)
0.172   memory lost to fragmentation
537.845 total actual memory usage
```

3.7.2. Verification Results

SPIN has generated an error trail which contains a sequence of events leading to a violation condition for our claim. Here, we use again XSPIN to perform a guided simulation using the generated error trail. This guided simulation provides us an *MSC* which makes the attack visible (See Figure 3.7[5]). The logical representation of this attack can also be examined in Figure 3.8 and Figure 3.9 for RREQ propagation, and RREP forwarding, respectively.

In [52], the authors state that with an *Active-y-x* attacker configuration, lengthening the route in the RREQ by adding other compromised nodes to the route is possible. In this attack, if there is a shorter route, the source terminal will prefer that one; thereby rendering the attacker powerless[6]. However, the attack that SPIN flagged is much more general and powerful; since it shows that the actual existing route can be shortened by the compromised nodes, which forces the initiator to prefer this route. This result shows that one of the targeted security properties of ARIADNE, the property that no intermediate node is able to remove any other nodes from the route-discovery process may not hold in an *Active-2-2* adversary environment. This might possibly be used for routing-disruption or some other purposes like traffic analysis etc.

[5]Please note that we have redrawn the MSC outputs that SPIN generated since the original ones have a low readability.

[6]However, it should be noted that if the aim of the attacker is to make the source terminal choose any other route than the route over which the attacker lies, this attack could also be useful.

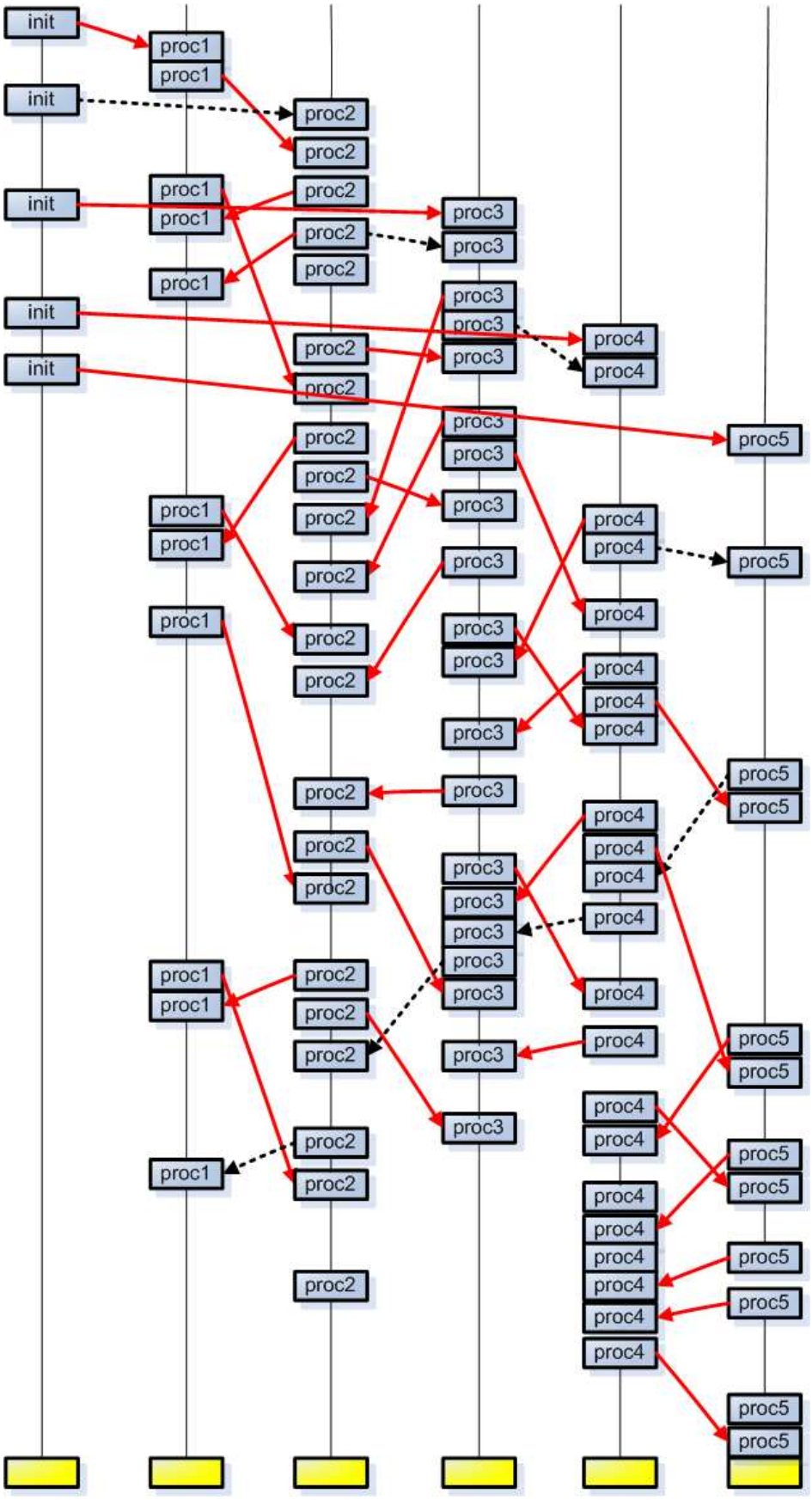

Figure 3.7. SPIN's error trail showing how a violation may occur.

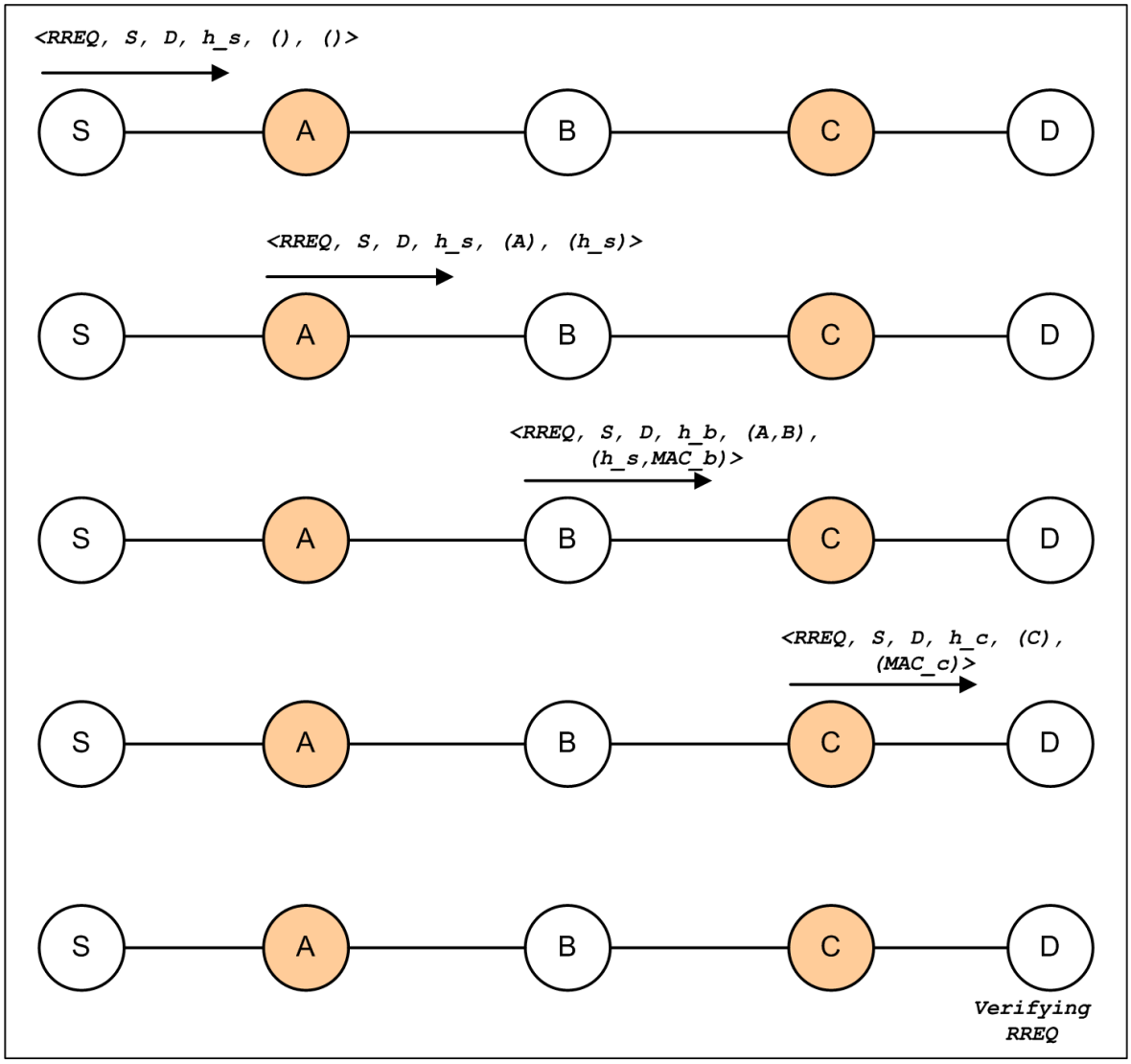

Figure 3.8. RREQ propagation phase representation of the error trail generated by SPIN, demonstrating a new attack on ARIADNE, which is of type *Active-2-2*.

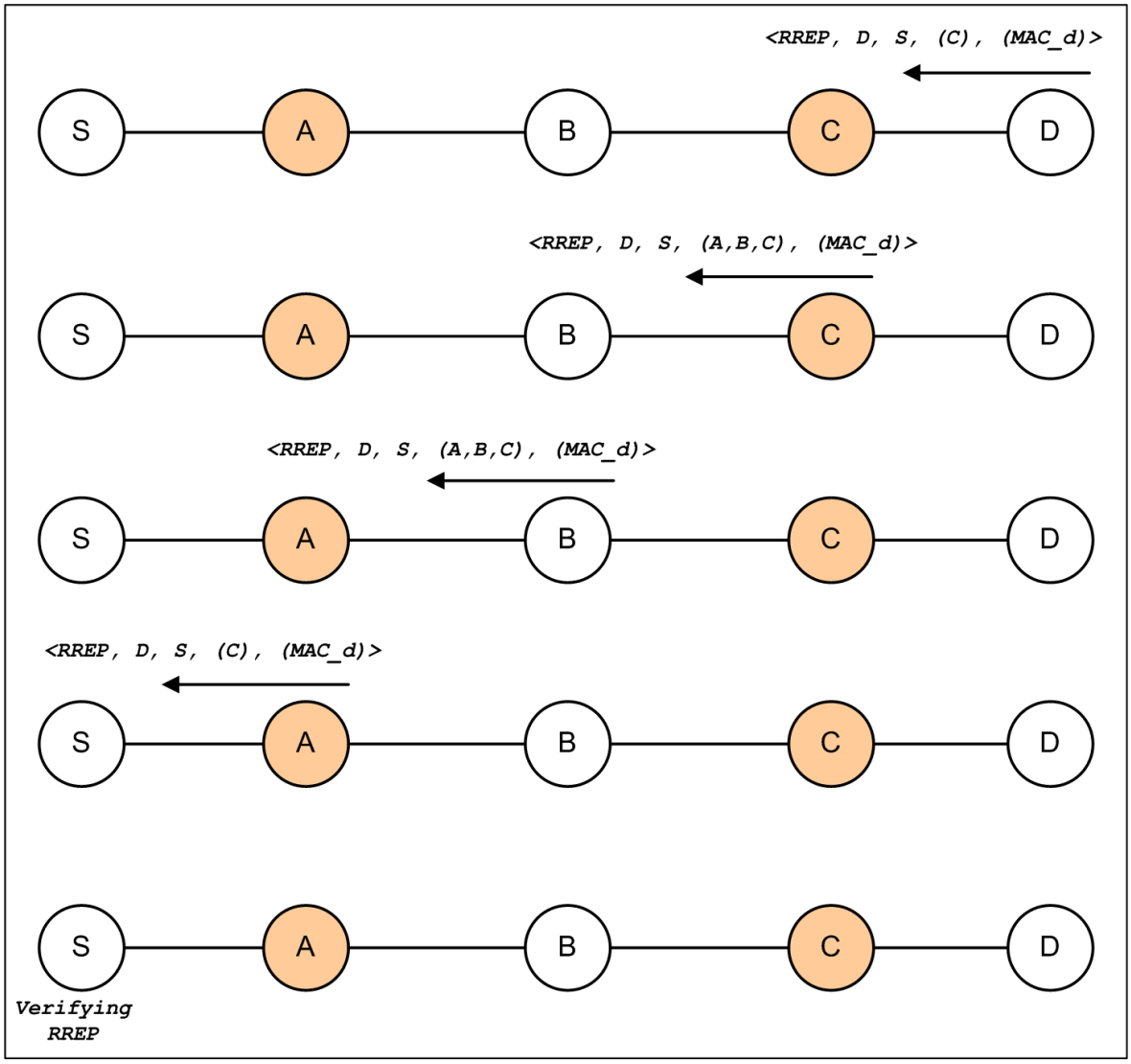

Figure 3.9. RREP forwarding phase representation of the error trail generated by SPIN, demonstrating a new attack on ARIADNE, which is of type *Active-2-2*.

4. CONCLUSIONS AND FUTURE DIRECTIONS

It is widely accepted that secure protocols need a proof of security before being acceptable. The main flavor of formal methods, which are defined to be mathematical techniques for the specification, development and verification of complex systems, is that it is possible to achieve provable correctness and reliability in any system design and to analyze a system for desired properties.

We used model checking as our formal verification technique that relies on building a finite model of a system and checking that a desired property holds in that model. We specified and executed a model checking approach for the secure ad hoc routing protocol ARIADNE using SPIN; which is a widely accepted public domain general-purpose model checking tool used for specification, simulation, validation and verification of asynchronous concurrent processes. Our work is the first model checking approach in the literature on security property verification of an ad hoc network routing protocol. The reason we have chosen ARIADNE is not only for the fact that it has very powerful security properties, but also its authors have introduced a new attacker classification scheme with the term *Active-y-x*; where y stands for the number of terminals that the attacker has compromised and x stands for the total number of terminals that the attacker owns within the network.

We built an ad hoc network model, along with the model of a legitimate node and two distinct compromised nodes; whose behaviors are similar to the ones that Buttyán *et al.* described in order to perform an *Active-1-2* attack on ARIADNE [1]. The authors in [1] also stated that checking for repeating identifiers in the RREP packet may cure this flaw. Our approach is distinct from theirs in the sense that our compromised nodes try to perform an *Active-2-2* attack; in which the subverted terminals can use two distinct identifiers legitimately. Therefore, this attack can not be cured by the same approach.

SPIN generated automata for our modeled processes, making the state space anal-

ysis possible. Our state space is around $2,22 \times 10^{14}$ states, which is huge. Therefore we used some of SPIN's compile-time directives to reduce the memory requirements. At last, we could find a violation to our claim after the verifier visited 673 million states in the state space.

The verification run gave us an error trail, i.e., a sequence of possible events in the protocol which leads to a violation of the property we are trying to verify. This result shows that one of the targeted security properties of ARIADNE, the property that *no intermediate node is able to remove any other nodes from the route-discovery process* may not hold in an *Active-2-2* adversary environment.

It should be noted that both of these attacks are not limited to a logically linear five-node ad hoc network; instead, having the adversaries placed anywhere in the network topology, the nodes between them can easily be removed from the *node-list* of any RREQ traversing that route. These types of attacks are very powerful since the actual existing route can be shortened by the compromised nodes, which forces the initiator to prefer this route. After that, the compromised nodes can manage the communication between the initiator and the destination.

This work is an extension of Buttyán's attack on ARIADNE, which uses the model-checker SPIN to flag the attack. We extended his attack from an *Active-1-2* environment to an *Active-2-2* environment. We did not target to perform a full verification of the protocol but to find a sequence of events leading to a violation of a security property. Therefore, for the sake of simplicity, we did not include some core characteristics of the ad hoc environment and the protocol in our model; such as mobility, and route-error type messages. For a full verification of the protocol, these aspects should also be included in the model; but with the right level of abstraction which does not sacrifice the functionality while keeping the model simple.

Furthermore, our attacker model consists of nodes which know how to behave in order to launch the specified attack. Instead, a generic attacker model whose sequence of behaviors are unpredictable even by the modeler of the attacker may have the chance of revealing yet undiscovered attacks on ARIADNE.

APPENDIX A: FINITE STATE AUTOMATA OF NETWORK NODES IN OUR MODEL

A.1. Legitimate Nodes

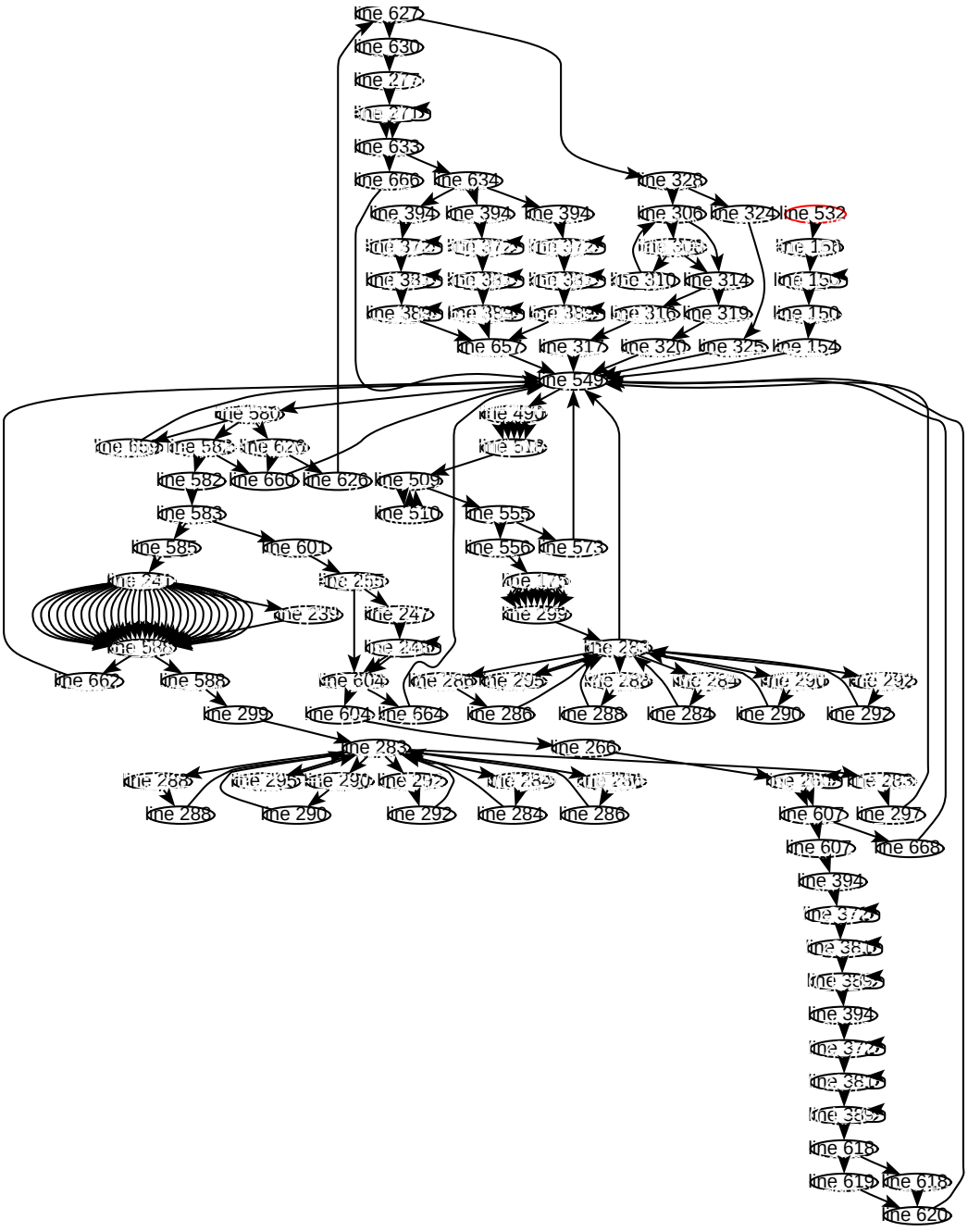

Figure A.1. SPIN automaton of a legitimate node in our model.

A.2. First Compromised Node

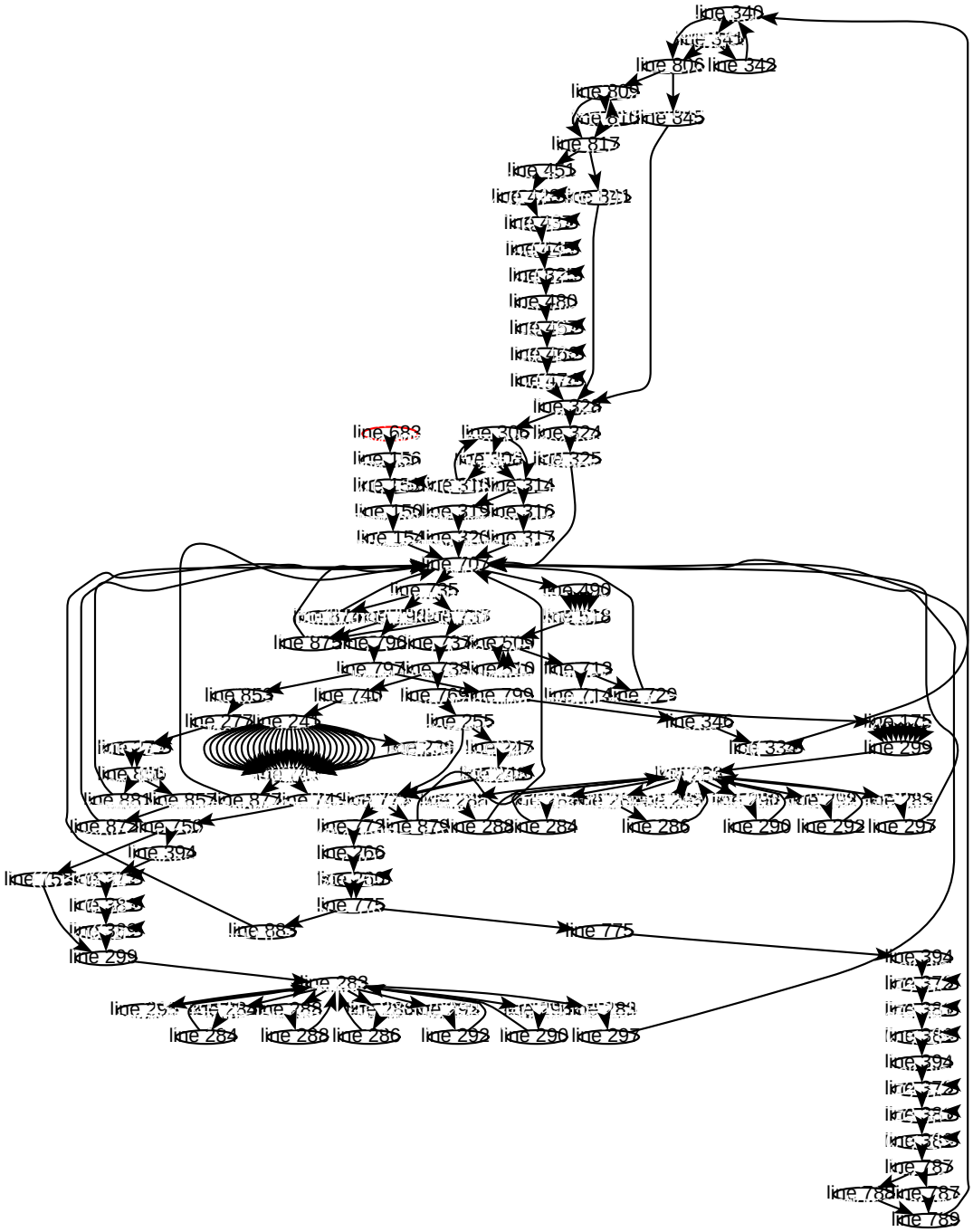

Figure A.2. SPIN automaton of the first compromised node in our model.

A.3. Second Compromised Node

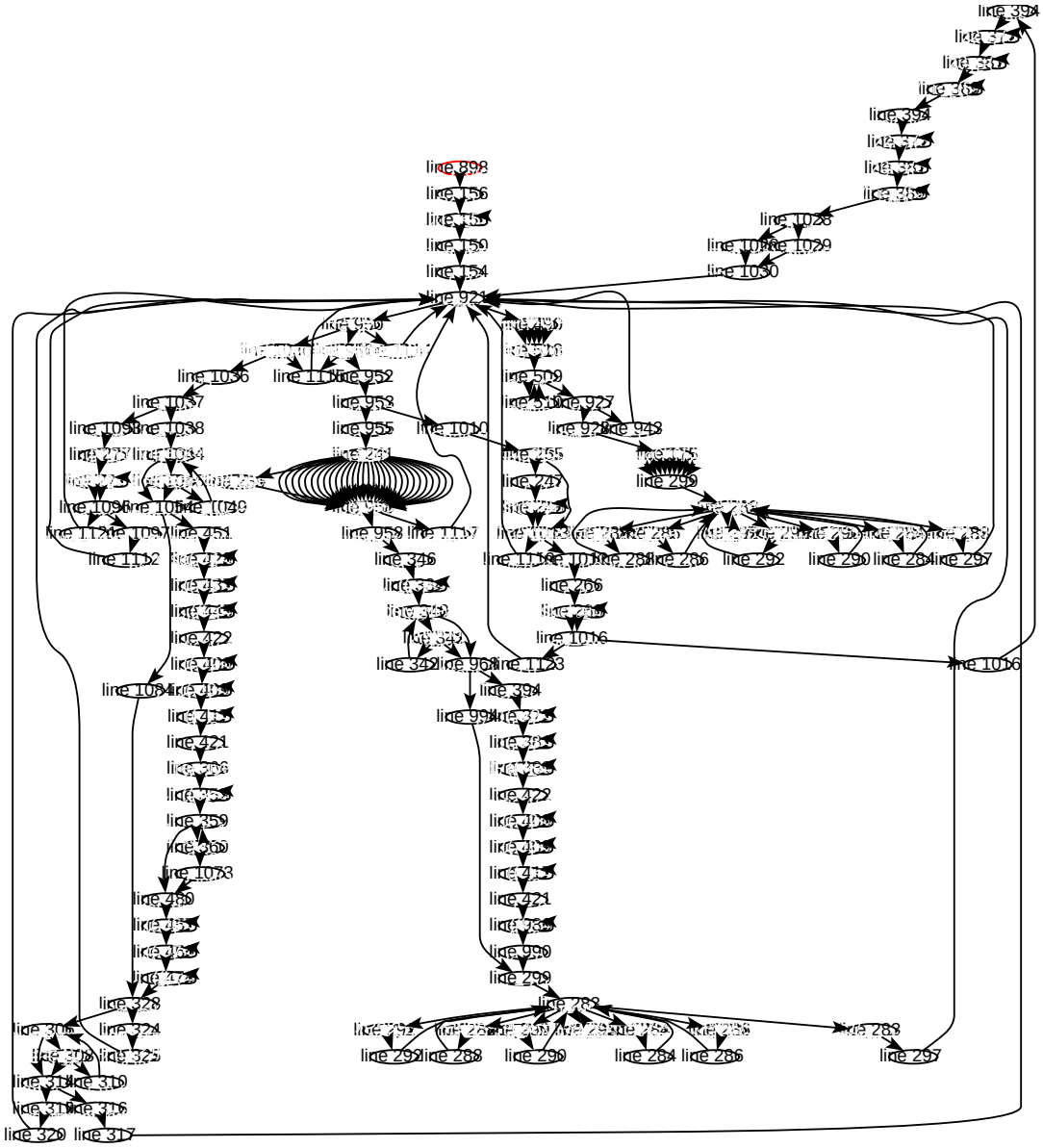

Figure A.3. SPIN automaton of the second compromised node in our model.

REFERENCES

1. L.Buttyán, G.Ács, I.Vajda, "Provably Secure On-demand Source Routing in Mobile Ad Hoc Networks", IEEE Transactions on Mobile Computing (to appear), 2005 (year of approvement).

2. Z.J.Haas, J.Deng, B.Liang, P.Papadimitratos, S.Sajama, "Wireless Ad Hoc Networks", Cornell University, 2002.

3. E.M.Belding-Royer, C.K.Toh, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks", IEEE Personal Communications Magazine, p.46-55, 1999.

4. C.Perkins, T.Watson, "Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for Mobile Computers", ACM SIGCOMM'94 Conference on Communications Architectures, 1994.

5. S.Murthy, J.Garcia-Luna-Aceves, "A Routing Protocol for Packet Radio Networks", Proceedings of the First Annual ACM International Conference on Mobile Computing and Networking, pp.86-95, 1995.

6. T.W.Chen, M.Gerla, "Global State Routing: A New Routing Scheme for ad hoc Wireless Networks", Proceedings of the IEEE ICC, 1998.

7. G.Pei, M. Gerla, T.W.Chen, "Fisheye State Routing in Mobile Ad Hoc Networks", ICDCS Workshop on Wireless Networks and Mobile Computing, 2000.

8. J.Garcia-Luna-Aceves, C.M.Spohn, "Source-Tree Routing in Wireless Networks", Proceedings of the Seventh Annual International Conference on Network Protocols, 1999.

9. S.Basagni, I.Chlamtac, V.Syrotivk, B.Woodward, "A Distance Effect Algorithm

for Mobility (DREAM)", Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'98), 1998.

10. K.Kasera, R.Ramanathan, "A Location Management Protocol for Hierarchically Organised Multihop Mobile Wireless Networks", Proceedings of the IEEE ICUPC'97, pp.158-162, 1997.

11. C.C.Chiang, "Routing in Clustered Multihop Mobile Wireless Networks with Fading Channel", Proceedings of IEEE SICON, pp.197-211, 1997.

12. G.Pei, M. Gerla, X.Hong, C.Chiang, "A Wireless Hierarchical Routing Protocol with Group Mobility", Proceedings of Wireless Communications and Networking, 1999.

13. T.H.Clausen, P.Jacquet, "Optimized Link State Routing Protocol (OLSR)", RFC 3626, 2003.

14. R.G.Ogier, F.L.Templin, M.G.Lewis, "Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)", RFC 3684, 2004.ITU-T, Recommendation Z.100 (11/99) Specification and Description Language (SDL)

15. D.B.Johnson, D.A.Maltz, Y.C.Hu, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)", Internet Draft, 2004.

16. C.E.Perkins, E.M.Belding-Royer, S.R.Das, "Ad Hoc On-Demand Distance Vector (AODV) Routing", RFC 3561, 2003.

17. M.Corson, A.Ephremides, "A Distributed Routing Algorithm For Mobile Wireless Networks", ACM/Baltzer Wireless Networks, Vol.1, No.1, pp.61-81, 1995.

18. J.Raju, J.Garcia-Luna-Aceves, "A New Approach to On-Demand Loop-Free Multipath Routing", Proceedings of the 8th Annual IEEE International Conference on Computer Communications and Networks (ICCCN), pp.522-527, 1999.

19. V.Park, M. Corson, "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks", Proceedings of INFOCOM, 1997.

20. C.Toh, "A Novel Distributed Routing Protocol to Support ad hoc Mobile Computing", IEEE 15th Annual International Phoenix Conference, pp.480-486, 1996.

21. R.Dube, C.Rais, K.Wang, S.Tripathi, "Signal Stability Based Adaptive Routing (SSA) for Ad Hoc Mobile Networks", IEEE Personal Communication, Vol.4, No.1, pp.36-45, 1997.

22. G.Aggelou, R.Tafazolli, "RDMAR: A Bandwidth-Efficient Routing Protocol for Mobile Ad Hoc Networks", ACM International Workshop on Wireless Mobile Multimedia (WoWMoM), pp.26-33, 1999.

23. Y.B.changed. Ko, N.Vaidya, "Location-Aided Routing (LAR) in Mobile Ad Hoc Networks", Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'98), 1998.

24. M.Gunes, U.Sorges, I.Bouazzi, "Ara - the Ant-Colony Based Routing Algorithm for Manets", ICPP workshop on Ad Hoc Networks (IWAHN 2002), pp.79-85, 2002.

25. W.Su, M.Gerla, "Ipv6 Flow Handoff in ad hoc Wireless Networks Using Mobility Prediction", IEEE Global Communications Conference, pp.271-275, 1999.

26. M.Jiang, J.Ji, Y.Tay, "Cluster-Based Routing Protocol", Internet Draft, draft-ietf-manet-cbrp-spec-01.txt, 1999.

27. Z.Hass, M.Pearlman, "Zone Routing Protocol for ad hoc Networks", Internet Draft, draft-ietf-manet-zrp-05.txt, 1999.

28. M.Joa-Ng, I.T.Lu, "A Peer-to-Peer Zone-Based Two-Level Link State Routing for Mobile Ad Hoc Networks", IEEE Journal on Selected Areas in Communications, Vol.17, No.8, pp.1415-1425, 1999.

29. S.C.Woo, S.Singh, "Scalable Routing Protocol for Ad Hoc Networks", Wireless Networks, Vol.7, No.5, pp.513-529, 2001.

30. S.Radhakrishnan, N.Rao, G.Racherla, C.Sekharan, S.Batsell, "DST - A Routing Protocol for Ad Hoc Networks Using Distributed Spanning Trees", IEEE Wireless Communications and Networking Conference, 1999.

31. N.Nikaein, H.Laboid, C.Bonnet, "Distributed dynamic routing algorithm (DDR) for mobile ad hoc networks", Proceedings of the First Annual Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC 2000), 2000.

32. S.R.Das, R.Castaneda, Y.Jiangtao, R.Sengupta, "Comparative Performance Evaluation of Routing Protocols for Mobile Ad Hoc Networks", IEEE International Conference on Computer Communications and Networks, pp.153-161, 1998.

33. S.J.Lee, M.Gerla, C.K.Toh, "A Simulation Study of Table-Driven and On-Demand Routing Protocols for Mobile Ad Hoc Networks. IEEE Networks, 13(4):48-54, 1999.

34. A.Barbir, S.Murphy, Y.Yang, "Generic Threats to Routing Protocols", Internet Draft, 2004, http://www.ietf.org/internet-drafts/draft-ietf-rpsec-routing-threats-06.txt.

35. J.-F.Raymond, "Traffic Analysis: Protocols, Attacks, Design Issues and Open Problems", Proc. Workshop on Design Issues in Anonymity and Unobservability, pp.7-26, 2000.

36. Y.C.Hu, A.Perrig, "A Survey of Secure Wireless Ad Hoc Routing", IEEE Security and Privacy, 2004.

37. D.Djenouri, L.Khelladi, "A Survey of Security Issues in Mobile Ad Hoc and Sensor Networks", IEEE Communications Surveys and Tutorials, Volume-7, No-4, 2005.

38. A.Patwardhan, J.Parker, A.Joshi, M.Iorga, T.Karygiannis, "Secure Routing and

Intrusion Detection in Ad Hoc Networks", Proceedings of the 3rd International Conference on Pervasive Computing and Communications, 2005.

39. K.Inkinen, "New Secure Routing in Ad Hoc Networks: Study and Evaluation of Proposed Schemes", HUT T-110.551 Seminar on Internetworking, 2004.

40. http://tools.ietf.org/html/draft-ietf-rpsec-routing-threats-02.

41. M.G.Zapata, N.Asokan, "Securing Ad Hoc Routing Protocols", ACM Workshop on Wireless Security (WiSe'02), 2002.

42. B.Wu, J.Chen, J.Wu, M.Cardei, "A Survey on Attacks and Countermeasures in Mobile Ad Hoc Networks", Wireless and Mobile Network Security, 2006.

43. F.Stajano, R.Anderson, "The Resurrecting Duckling: Security Issues for ad hoc Wireless Networks", Proc. 7th Int'l Workshop on Security Protocols, Lecture Notes in Computer Science 1796, Springer-Verlag, pp.172-194, 1999.

44. D.Balfanz *et al.*, "Talking to Strangers: Authentication in ad hoc Wireless Networks", Proc. Symp. Network and Distributed Systems Security (NDSS 2002), Internet Society, pp.23-35, 2002.

45. S.Zhu, S.Xu, S.Setia, S.Jajodia, "Establishing Pair-Wise Keys for Secure Communication in Ad Hoc Networks: A Probabilistic Approach", IEEE International Conference on Network Protocols, 2003.

46. L.Zhou, Z.Haas, "Securing Ad Hoc Networks", IEEE Network Magazine Special Issue on Network Security, Vol.13, No.6, 1999.

47. G.Montenegro, C.Castelluccia, "Statistically Unique and Cryptographically Verifiable (SUCV) Identifiers and Addresses", Proceedings of the 9th Annual Network and Distributed System Security Symposium (NDSS), 2002.

48. S.Ćapkun, J.P.Hubaux, "BISS: Building Secure Routing out of an Incomplete

Set of Security Associations", Proceedings of the Wireless Security Workshop (WISE'03), 2003.

49. J.Kong, P.Zerfos, H.Luo, S.Lu, L.Zhang, "Providing Robust and Ubiquitous Security Support for Mobile Ad Hoc Networks", In Proceedings of the 9th International Conference on Network Protocols (ICNP), 2001.

50. N.Asokan and P.Ginzboorg, "Key Agreement in Ad Hoc Networks", Computer Communications, 23:1627-1637, 2000.

51. S.Ćapkun, L.Buttyán, J.P.Hubaux, "Self-Organized Public-Key Management for Mobile Ad Hoc Networks", IEEE Transactions on Mobile Computing, 2(1), January-March 2003.

52. Y.C.Hu, D.B.Johnson, A.Perrig, "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks", MOBICOM'02, 2002.

53. Y.C.Hu, D.B.Johnson, A.Perrig, "SEAD: Secure Effcient Distance Vector Routing for Mobile Wireless Ad Hoc Networks", Ad Hoc Networks, Vol.1, pp.175-192, 2003.

54. S.Gupte, M.Singhal, "Secure Routing in Mobile Wireless Ad Hoc Networks", Ad Hoc Networks 1, 151-174, 2003.

55. K.Sanzgiri, B.Dahill, B.N.Levine, C.Shields, E.M.Belding-Royer, "A Secure Routing Protocol for Ad Hoc Networks", Proceedings of 2002 IEEE International Conference on Network Protocols (ICNP), 2002.

56. P.Papadimitratos, Z.J.Haas, "Secure Routing for Mobile Ad Hoc Networks", Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002), 2002.

57. N.Padoy, "Secure Routing in Ad Hoc Networks", Seminar Ad Hoc Networking, in http://www13.informatik.tu-muenchen.de/lehre/seminare/WS0304/UB-

hs/npadoy-adhoc-paper.ps, 2003.

58. P.Papadimitratos, Z.J.Haas, "Secure Message Transmission in Mobile Ad Hoc Networks", Elsevier Ad Hoc Networks Journal, Elsevier, vol.1, no.1, pp.193-209, 2003.

59. S.Yi, P.Naldurg, R.Kravets, "A Security-Aware Ad Hoc Routing Protocol for Wireless Networks", In The 6th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2002), 2002.

60. P.Papadimitratos, Z.J.Haas, "Secure Link State Routing for Mobile Ad Hoc Networks", Proceedings of the IEEE Workshop on Security and Assurance in Ad Hoc Networks in conjunction with the 2003 International Symposium on Applications and the Internet, 2003.

61. B.Awerbuch, D.Holmer, C.Nita-Rotaru,H.Rubens, "An On-Demand Secure Routing Protocol Resilent to Byzantine Failures", ACM Workshop on Wireless Security (WiSe), 2002.

62. T.Wan, E.Kranakis, P.C.Oorschot, "Securing the Destination-Sequenced Distance Vector Routing Protocol (S-DSDV)", Proceedings of the 6th International Conference on Information and Communications Security, 2004.

63. U.Lu, B.Pooch, "Cooperative Security-Enforcement Routing in Mobile Ad Hoc Networks", Mobile and Wireless Communications Network, 4th International Workshop on, Vol., Iss., pages 157-161, 2002.

64. S.Eichler, C.Roman, "Challenges of Secure Routing in MANETs: A Simulative Approach using AODV-SEC", Proceedings of the 3rd IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS), 2006.

65. R. Ramanujan, A. Ahamad, and K. Thurber, "Techniques for Intrusion Resistant Ad Hoc Routing Algorithms (TIARA)", Proc. Military Communications Conf.

(MILCOM 2000), pp.660-664, 2000.

66. S.Carter, A.Yasinsac "Secure Position Aided Ad Hoc Routing Protocol", Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN02), 2002.

67. T.Clausen, C.Adjih, P.Jacquet, A.Laouiti, A.Muhlethaler, D.Raffo, "Securing the OLSR Protocol", Proceeding of IFIP Med-Hoc-Net, 2003.

68. S. Buchegger, and J.-Y. Le Boudec, "Performance Analysis of the CONFIDANT Protocol (Cooperation Of Nodes: Fairness In Dynamic Ad Hoc Networks)", Proc. 3rd Symp. Mobile Ad Hoc Networking and Computing (MobiHoc 2002), pp.226-236, 2002.

69. P.G.Argyroudis, D.O'Mahony, "Secure Routing for Mobile Ad Hoc Networks", IEEE Communications Surveys and Tutorials, Volume-7, pp.2-21, 2005.

70. S.Marti, T.J.Giuli, K.Lai, M.Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks", Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking, 2000.

71. Y.-C.Hu, A.Perrig, D.B.Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks", Proc. 22nd Ann. Joint Conf. IEEE Computer and Communications Societies (INFOCOM), 2003.

72. L.Hu, D.Evans, "Using Directional Antennas to Prevent Wormhole Attacks", Proc. of Networks and Distributed System Security Symposium (NDSS), 2004.

73. S.Capkun, L.Buttyan, J.Hubaux, "Sector: Secure Tracking of Node Encounters in Multi-hop Wireless Networks", Proc. of the ACM Workshop on Security of Ad Hoc and Sensor Networks, 2003.

74. $http://en.wikipedia.org/wiki/Formal_methods$.

75. Anthony Hall, "Seven Myths of Formal Methods", IEEE Software, V.7, pp.11-19, 1990.

76. F.Babich, L.Deotto, "Formal Methods for Specification and Analysis of Communication Protocols", IEEE Communications Surveys, 2002.

77. ITU-T, Recommendation Z.100 (11/99) Specification and Description Language (SDL).

78. $http://en.wikipedia.org/wiki/Specification_and_Description_Language$.

79. $http://www.dcc.ufmg.br/\ coelho/jade.html$.

80. $http://www.informatik.hu-berlin.de/Themen/SITE/5$.

81. $http://www.estelle.org/$.

82. T.Bolognesi, E.Brinskma, "Introduction to the ISO Specification Language LOTOS", Computer Networks and ISDN Systems, vol.14, pp.92100, 1987.

83. K.G.Larsen, P.Pettersson, W.Li, "UPPAAL in a Nutshell", Springer Intl. J. Software Tools for Technology Transfer, vol. 1+2, 1997.

84. $http://www.avispa-project.org/$.

85. $http://en.wikipedia.org/wiki/Formal_verification$.

86. E.M.Clarke, J.M.Wing, "Formal Methods: State of the Art and Future Directions", ACM Computing Surveys, V.28, pp.626-643, 1996.

87. E.M.Clarke, E.A.Emerson, "Synthesis of Sychronization Skeletons for Branching Time Temporal Logic", Logic of Programs: Workshop, Yorktown Heights, NY, Volume 131 of Lecture Notes in Computer Science, 1981.

88. J.Queille, J.Sifakis, "Specification and Verification of Concurrent Systems in CAESAR", Proc. of Fifth ISP, 1982.

89. Ronan de Renesse, "Wireless Adaptive Routing Protocol(WARP) Verification Using SPIN Model Checker", MSC by Research in Telecommunications, King's College London, September 2003.

90. R.E.Bryant, "Graph Based Algorithms for Boolean Function Manipulation", IEEE Transactions on Computers C-35, 1986.

91. R.Alur, R.K.Brayton, T.A.Henzinger, S.Qadeer S.K.Rajamani, "Partial-Order Reduction in Symbolic State-Space Exploration", Proc. Intl. Conf. CAV'97, 1997.

92. R.P.Kurshan, "Computer-Aided Verification of Coordinating Processes", Princeton University Press, 1994.

93. K.G.Larsen, P.Petterson, W.Yi, "Compositional and Symbolic Model Checking of Real-Time Systems", Proc. 16th IEEE Real-Time Systems Symposium, pp.76-87, 1995.

94. W.Elseaidy, R.Cleaveland, J.Baugh, "Modeling and Verifying Active Structural Control Systems", Proceedings of the 1994 Real-Time Systems Symposium, 1996.

95. G.J.Holzmann, "The Spin Model Checker: Primer and Reference Manual", Addison-Wesley, 2003.

96. S.Yang, J.Baras. "Modeling Vulnerabilities of Ad Hoc Routing Protocols", Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks, October 2003.

97. A.Pnueli, The Temporal Logic of Programs", Proc. 18th IEEE Symp. Foundations of Computer Science, pp.46-57, 1977.

98. G.Lowe, "Breaking and Fixing the Needham-Schroeder Public-Key Protocol Us-

ing FDR", Tools and Algorithms for the Construction and Analysis of Systems, volume 1055 of Lecture Notes in Computer Science, pp. 147166, Springer-Verlag, 1996.

99. J.C.Mitchell, M.Mitchell, U.Stern, "Automated Analysis of Cryptographic Protocols Using Murφ", Proceedings of the 1997 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1997.

100. J.Mitchell, V.Shmatikov, U.Stern, Finite-State Analysis of SSL 3.0, 7th USENIX Security Symposium, 1998.

101. M.Burrows, M.Abadi, R.Needham, "A Logic of Authentication", ACM Transactions on Computer Systems, pp. 18-36, 1990.

102. C.Meadows, "A Procedure for Verifying Security Against Type Confusion Attacks", Proceedings of the 16th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, June 2003.

103. D.X.Song, S.Berezin, A.Perrig, "Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis", Journal of Computer Security, v. 9, pp. 47-74, 2001.

104. G.Bella, L.C.Paulson, "Using Isabelle to Prove Properties of the Kerberos Authentication System", DIMACS Workshop on Design and Formal Verification of Security Protocols, 1997.

105. L.C.Paulson, "Inductive Analysis of the Internet Protocol TLS", ACM Transactions on Information and System Security 2 (3), pp. 332-351, 1999.

106. L.C.Paulson, "Natural Deduction as Higher-Order Resolution", Journal of Logic Programming (3), pp. 237-258, 1986.

107. G.Bella, F.Massacci, L.C.Paulson, "An overview of the verification of SET", International Journal of Information Security, in press.

108. G.Bella, C. Longo, L.C.Paulson, "Verifying Second-Level Security Protocols", Theorem Proving in Higher Order Logics (Springer LNCS 2758), pp. 352-366, 2003.

109. B.Donovan, P.Norris, G.Lowe, "Analyzing a Library of Security Protocols Using Casper and FDR", Proceedings of the Workshop on Formal Methods and Security Protocols, 1999.

110. Y.Chevalier, L.Vigneron, "Automated Unbounded Verification of Security Protocols", Proc. CAV'02, LNCS 2404. Springer-Verlag, 2002.

111. E.M.Clarke, S.Jha, W.Marrero, "Verifying Security Protocols with Brutus", ACM Transactions on Software Engineering and Methodology, Vol. 9, No. 4, October 2000.

112. M.Bellare, J.A.Garay, R.Hauser, A.Herzberg, H.Krawczyk, M.Steiner, G.Tsudik, M.Waidner, "iKP - A Family of Secure Electronic Payment Protocols, 1st USENIX Workshop on Electronic Commerce, pp. 89-106, 1995.

113. T.Coffey, R.Dojen, T.Flanagan, "Formal Verification: An Imperative Step in the Design of Security Protocols", Computer Networks (43), pp. 601-618, 2003.

114. M.J. Beller, L.-F. Chang, Y. Yacobi, Privacy and Authentication on a Portable Communications System, IEEE Journal on Selected Areas in Communications 11 (6), pp. 821-829, 1993.

115. D.Basin, K.Miyazaki, K.Takaragi, "A Formal Analysis of a Digital Signature Architecture", Integrity and Internal Control in Information Systems, IV, Kluwer Academic Publishers, pp. 31-48, 2004.

116. D.Basin, S.Modersheim, L.Vigano, "OFMC: A Symbolic Model Checker for Security Protocols", International Journal of Information Security, 2004.

117. G.Steel, A.Bundy, "Attacking Group Multicast Key Management Protocols Using

CORAL", Electronic Notes in Theoretical Computer Science (ENTCS), 125(1), pp. 125-144, 2005.

118. C.Caleiro, L.Vigan, D.Basin, "Metareasoning About Security Protocols Using Distributed Temporal Logic", Electronic Notes in Theoretical Computer Science 125, pp. 67-89, 2005.

119. P.Maggi, R.Sisto, "Using SPIN to Verify Security Properties of Cryptographic Protocols", Proceedings of the 9th SPIN Workshop, Grenoble, France, 2002.

120. A.S.Khan, M.Mukund, S.P.Suresh, "Generic Verification of Security Protocols", Proceedings of the 12th SPIN Workshop on Model Checking Software San Francisco, USA, 2005.

121. T.Lauschner, A.Macedo, S.Campos, "Formal Verification and Analysis of a Routing Protocol for Ad Hoc Networks", *http* : *//www.dcc.ufam.edu.br/ tanara/artigo.html*, July 1997.

122. K.Bhargavan, D.Obradovic, C.A.Gunter, "Formal Verification of Standards for Distance Vector Routing Protocols", Journal of the ACM (JACM), 49(4):538 576, 2002.

123. D.Obradovic, "Formal Analysis of Routing Protocols", PhD thesis, University of Pennsylvania, 200.

124. J.Marshall, "An Analysis of the Secure Routing Protocol for Mobile Ad Hoc Network Route Discovery: Using Intuitive Reasoning and Formal Verification to Identify Flaws", MSc thesis, Department of Computer Science, Florida State University, April 2003.

125. I.Zakiuddin, M.Goldsmith, P.Whittaker, P.Gardiner, "A Methodology for Model Checking Ad Hoc Networks", Proceedings of the 10th SPIN Workshop, 2003.

126. X.Liu, J.Wang, "Formal Verification of Ad Hoc On-demand Distance Vector

(AODV) Protocol using Cadence SMV", CPSC513 course project, University of British Columbia, 2004.

127. S.Chiyangwa, M.Kwiatkowska, "A Timing Analysis of AODV", Proceedings of the Formal Methods for Open Object-Based Distributed Systems: 7th IFIP WG 6.1 International Conference, FMOODS 2005, June 15-17, 2005.

128. S.Nanz, C.Hankin, "Formal Security Analysis for Ad Hoc Networks", Proceedings of the 2004 Workshop on Views on Designing Complex Architectures (VODCA'04), 2004.

129. S.Nanz, C.Hankin, "Static Analysis of Routing Protocols for Ad Hoc Networks", Proceedings of the 2004 ACM SIGPLAN and IFIP WG 1.7 Workshop on Issues in the Theory of Security (WITS'04), pp. 141-152, 2004.

130. L.Buttyán, I.Vajda, "Towards Provable Security for Ad Hoc Routing Protocols", Proceedings of ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'04), 2004.

131. T.Ruys, "Towards Effective Model Checking", Phd Thesis, Centre for Telematics and Information Technology, 2001.

132. D.Dolev, A.Yao, "On the Security of Public Key Protocols", Proc. IEEE Symp. on Foundations of Computer Science, pages 350-357, 1981.