Secure Routing in Ad Hoc Networks

Pedro Brandão, Susana Sargento, Sérgio Crisóstomo, Rui Prior
Department of Computer Science & LIACC
Faculty of Sciences, University of Porto
Rua do Campo Alegre, 823, 4150-180 Porto, Portugal
{pbrandao, ssargento, slc, rprior}@dcc.fc.up.pt

May 2004

Abstract

Ad Hoc networks dramatically increase the network security concerns. This paper presents a security protocol for ad hoc networks, denoted AD hoc SEcure Routing (ADSER), that copes with the majority of the security issues raised by the incremental deployment of ad hoc networks. This protocol copes with message integrity and signing, encryption of information and key distribution, with a low computational complexity in the majority of tasks performed. ADSER takes as a baseline some current security protocols and addresses the secure routing concerns of both source routing and reactive next hop protocols. This protocol is able to mitigate eavesdropping through the encryption of data, identity problems through signatures, trust issues through prevention of wrong route advertisements, and replays through sequence number and unique values generation.

1 Introduction

Nowadays, driven by the increasing users' requirement to be connected to the Internet every-time and everywhere, there is a vast amount of research in the area of ad hoc networks. Their main objective is to enable the autonomous creation of communication channels between mobile devices. These channels shall adapt to highly dynamical network configurations, with nodes joining and leaving the network. The node's configuration must take into account that the number of nodes in the network is not known, there are no special network entities (no central servers expected), and that no user interaction for the configuration of network connections is expected.

When network topology does not allow communications to be performed point to point, nodes are required to cooperate to forward packets. This leads to many of the security problems associated with ad hoc networks. Nodes that are asked to forward packets may not be inherently 'good', and knowledge of 'goodness' is difficult to obtain, since nodes are not known 'a priori'. Another problem related to security is that all nodes in the same radio range share the same communication medium. This way, it is very simple to listen to other nodes communications. Moreover, most of the nodes in ad hoc networks are low powered, in both processing and battery capacity. Therefore, all tasks performed need to have low complexity.

The inherent characteristics of ad hoc networks lead to the following security concerns. Eavesdropping is the possibility to 'hear' the data traversing the shared medium. Denial of Service (DoS) attacks can be performed by flooding the nodes with more requests than they can handle, or flooding the medium (with bogus data), invalidating any communication. Identity problems are not specific of this type of network but their solution is highly dependent on the architecture/protocols used. We need to guarantee that nodes are who they pretend to be. Trust issues are related to the information nodes advertise, for example, routing information. In ad hoc networks we need to prevent the advertisement of wrong routes. Selfishness is not a security concern, but may prejudice the network performance of nodes, since 'bad' ones can decide to selectively forward only some packets. Finally, in replay attacks, nodes retransmit packets sent previously by other nodes, hoping to replay the actions to their profit.

The level of security demanded determines the restrictions of the security architecture. For example, in a military ad hoc network there is a mandatory requirement to exclude non identified nodes from the network, and eavesdropping on the network is not even a possibility. In this scenario, even network existence must be concealed. However, in civilian networks, one can even question whether to remove a 'bad' node from our usable nodes, or to keep it to contact an, otherwise, unreachable node. This is merely an example of solutions that are feasible to a certain level of security, but are not even an option to other levels.

In this paper we analyze some security protocols and mechanisms applied to routing in ad hoc networks, in terms of their efficiency and complexity. We realize that current proposals only address some security issues, leaving other security concerns unattended. With the objective of designing a security protocol for ad hoc routing that copes with the majority of the security issues, we present a protocol, denoted AD hoc SEcure Routing (ADSER), that copes with message integrity and signing, encryption of information and key distribution, with a low computational complexity in the majority of tasks performed. ADSER takes as a baseline some current security protocols. The routing protocols considered in the description of the security mechanisms are source routing protocols and reactive next hop protocols: Dynamic Source Roting (DSR) [1] and Ad hoc On demand Distance Vector routing (AODV) [2] will be used as examples.

The paper is organized as follows. In section 2 we address several proposals for secure routing in ad hoc networks and evaluate them in terms of efficiency and computational complexity. The proposals that will be used as a baseline of the secure routing protocol, are described in more detail in this section. In section 3 we present the security protocol ADSER, addressing its most relevant phases in source routing protocols, namely: node joining the network in 3.1, getting public keys in 3.2 and route discovery in 3.3. An extension of ADSER to reactive next hop protocols is presented in section 4. Finally, section 5 addresses some thoughts on the protocol evaluation and section 6 presents the most relevant conclusions and future work.

2 Protocols for Secure Routing in Ad Hoc Networks

Several security mechanisms, aiming at providing secure routing in ad hoc networks, have been proposed in the literature. Some of the mechanisms are more concerned with ensuring and enforcing good behavior from the nodes in the network, while others introduce cryptographic schemes to guarantee data integrity, confidentiality and signing. Here we present some important approaches and qualitatively analyze them in terms of efficiency and computational complexity.

The proposals presented in [3] and [4] measure the behavior of nodes. In these protocols, nodes watch over each other to see if packets are forwarded correctly. As it is possible to listen promiscuously to the medium, nodes verify that the next hop node forwards their packets correctly. If some node is not correctly forwarding the packets, it will be avoided by the routing protocols. Both these proposals suffer when they encounter collision problems, transmission power control or colluding nodes. Moreover, load processing at a node is very high, because it is required to store packets, listen to communications not addressed to itself and make packets' comparisons.

The mechanism proposed in [5] aims at providing a path, from source to destination, with nodes only from a specified security level. Nodes belong to a specific level by holding a symmetric encryption key for that level; thus encryption and signing is performed between nodes in the same level. This protocol tries to establish different security groups in the network, where routing messages are secured by encryption and signing. Its downside is that there is high processing load at each node, and key distribution is not addressed.

SAODV (Secure AODV) [6] is an extension of the AODV routing protocol with security mechanisms (refer to Appendix B for more details in AODV). This extension provides signing of Route REQuest (RREQ) and Route REPly (RREP) messages, so senders and receivers can ascertain that routing messages are really sent by the source address indicated in the routing message. SAODV also provides truthful metrics, with hash values correlating to the hop count value. The problem of SAODV is that it does neither address the encryption of messages, nor signatures of intermediate nodes. However, processing capacity is alleviated through the use of less demanding operations.

In [7], the Secure Routing Protocol is introduced. SRP defines a source routing protocol that relies on a security association established between the source and destination nodes. This association is made through a shared symmetric key, that is used to verify and validate routing messages. Verification of the reply message, that entails the path, asserts the veracity of the path found. The computational effort related to cryptographic operations is only done at the destination and source nodes. Message identifiers are generated in a non-predictable way to mitigate replays. Colluding nodes can attack the protocol (as is described in the proposal). Key distribution is not addressed.

The On-Demand Secure Routing Protocol Resilient to Byzantine Failures (ODSBR) [8] detects link fault locations, using a reactive approach, and adds cryptographic schemes to guarantee data integrity, confidentiality and signing. Ariadne [9] and Efficient Security Mechanisms for Routing Protocols (ESMRP) [10] aim at lowering computational effort by using one way functions and hash chains to secure respectively, DSR and distance vector protocols. Finally, the Self Securing Ad hoc Wireless Networks (SSAWN) [11] addresses the key distribution problem, proposing the definition of a distributed Certificate Authority (CA) using threshold secret sharing [12]. These protocols, ODSBR, Ariadne, ESMRP and SSAWN, will be used as a baseline of the secure routing protocol presented in this paper. Therefore, with the objective of providing a background on the baseline of our mechanisms, we will address these protocols in more detail in the following sub-sections.

2.1 On-Demand Secure Byzantine Routing Protocol

The aim of this protocol is to introduce a routing algorithm that is able to cope with Byzantine failures (catching the 'bad' nodes). Therefore, it provides methods for encryption and signing of data, detection of faulty links, and route discovery based on a metric that weights the faultiness of links. This proposal stems from the principle that only source and destination are to be trusted. Information in data packets is encrypted with a shared key between source and destination. Every node has a list of link weights that measure the expected reliability of every link known by the node. A heavy weight means a small reliability; this metric is used in the route discovery process.

The algorithm is divided in two phases: route discovery and fault detection. All cryptographic operations are performed using shared keys.

Route discovery follows the principles of on demand protocols (DSR is taken as example; refer to Appendix B for more details): the source broadcasts a RREQ and waits for a RREP. The RREQ message is signed and carries a sequence number. Each node checks that the message comes from an authorized node, checking the signature, and then signs and forwards the RREQ. When the destination node receives the request, it generates a signed RREP (including the sequence number) with an empty node list. This list is broadcasted by each node in the return path. Therefore, when a node receives a RREP message it calculates the total cost so far, using its internal weight list and the nodes list from the message. If the total computed cost is lower than the previous one, or if this is the first answer seen from source and destination with that sequence number, the node checks the signature of all nodes (verifying the traveled path), adds itself to the path, signs the message and broadcasts it. If the cost is higher or equal, or if the signatures are not correct, the message is dropped. When the source node receives the RREP it uses the same algorithm (except broadcasting the message), and updates its path list accordingly. The source node can then use the 'lighter' path to the destination.

The signature check performed after the cost calculation does not prevent a malicious node from altering the nodes list, and thus increase the cost of the path. The advantage of the current approach is resource savings, as the node only computes signatures if the cost is lower than the previous one. However, with this approach, it is also possible to delete nodes from the end of the list. It suffices to remove them from the list and remove their signatures. The authors mentioned in their proposal that the detection algorithm (portrayed below) will detect this problem if the virtual paths formed loose packets' acknowledges.

Fault detection serves as input for the weight list. It is performed using acknowledgements for data packets. The procedure is as follows: the source node encrypts the data for the destination and adds a packet counter (which identifies the packet) to the information to be sent. The destination node shall send an acknowledgement for each data packet received. A threshold for non-acknowledged packets exists. When the number of non-acknowledges exceeds this threshold the fault detection mechanism is triggered. Fault detection uses probing to detect the link failure. Using a list of nodes that are to be probed, a binary search on the path is performed. The list is added to each data packet, and a new node is added to the list each time the threshold is exceeded, until the two edge nodes of the faulty link are detected. Each node in this list shall acknowledge the packet (if it hears it), so if the fault on a link is temporary, previous non-answering nodes will acknowledge the packet. As can be seen in figure 1, node 4 is requested to send an acknowledge after the first, second and third failure (until the probable faulty link is discovered). Node 3 is only added to the probe list after the third failure.

The probe list is '*onion*' encrypted, which means that the information is encrypted for each node. For example, when requesting acknowledges from nodes 2, 3 and 4, this list is the result of the concatenation of the following encryptions (using the shared key between S and 2, S and 3, S and 4): List $= Enc_{S2}(2 \mid Enc_{S3}(3 \mid Enc_{S4}(4)))$, where | denotes concatenation of information and Enc the encryption procedure. Each node in the probe list has to decrypt the list before sending it to the next node in the path. This way a malicious node cannot change the probe list, only decrypt its onion layer.

The proposed protocol also adds constructs so that each probed node can check that data is not modified and prevent malicious nodes from dropping acknowledges.

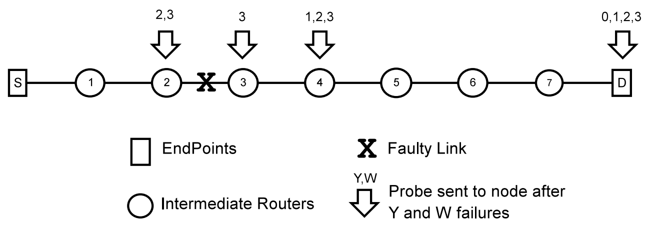

Figure 1: Fault detection on ODSRB

In this proposal, signing and encryption are addressed, as long as keys are distributed (which is not discussed); this addresses eavesdropping and identity problems. Faulty links can be detected, and an algorithm to provide protection of route discovery is portrayed, which accounts for trust issues. However, node removal from the path list is possible. Nodes have to perform multiple signature checks before sending the packet (although only when a path is better). Moreover, the computational complexity of the probing process is very high.

2.2 Ariadne and Efficient Security Mechanisms for Routing Protocols

Through the use of one-way functions the authors of Ariadne and ESMRP aim to provide identification, proof of metric correctness and correct paths with route discoveries, using low computational overheads. Ariadne addresses source routing protocols and ESMRP deals with a generic distance vector protocol.

In Ariadne the authors introduce security in the source routing protocol DSR. Its objective is to protect the path acquired in the route discovery process. It is assumed that a shared key is already established between source and destination nodes. For the calculation of Message Authentication Codes (MACs), used to provide message integrity, the nodes use digital signatures, shared keys or keys with a limited time to live generated using hash functions (TESLA [13]) (see the Appendix A for information on hash functions).

In route discovery, the RREQ carries the identification of the destination, of the source and the identification of the request. The source calculates a MAC with the shared key. The request is then broadcasted.

Each node that receives the request checks if this is a new request; if not the message is dropped. Otherwise, node i: (1) adds itself to a nodes list; (2) calculates a new hash value using its identification (ID) and the previous hash value from the previous node (i-1), using a one way function ($H_i = F(ID_i|H_{i-1})$); and (3) calculates MAC of the message with the new list. The message is then broadcasted with the nodes list, new hash value, and a MACs list (containing the MACs made so far in the path). The hash value and the MAC prevents the removal of nodes from the nodes list, as in the destination node H_J must be $F(ID_J| F(ID_{J-1}| F(ID_{J-2}|\ldots H_0)\ldots)))$, and each MAC from the MACs list must be valid (H_0 is the MAC from the source node using the shared key). Complete node removal, except for the source, is however possible, as H_0 is accessible to every node hearing the request, enabling the generation of $H_1 = F(ID_1|H_0)$.

When the destination node receives the request it checks the hash value as described. It also checks the MAC from the source, using the shared key. The destination then issues a reply sending the full path and the MACs from the nodes list. A MAC of the entire message using the shared key is also added. The reply is now unicast to the reverse path. When the sender receives the reply it checks the validity of the MAC from the destination and of each MAC in the nodes list.

Ariadne also provides security in routing errors and proposes a path reliability assessment scheme.

ESMRP uses the same concepts of Ariadne, being also based in hash functions. It adds identification and metric correctness to distance vector protocols.

The protocol has a sequence number and a hop metric. It ensures that a node does not decrease the metric or increase the sequence number. To achieve this, it generates a hash chain as shown in figure 2, where each hash value represents a given metric associated with a given sequence number. The hash values are derived as described in A, being $H_J = F(H_{J-1})$. Each row in the figure denotes the hash values used for a specific sequence number. The columns represent different metrics. Therefore, M is the maximum hop metric value and that hash chain can represent at most S different sequence numbers. These values are globally known. $H_{S \cdot M}$ is also disclosed by the requesting node with a signature asserting its origin.

When the node issues a RREQ, it adds a hash value of the first column, corresponding to its current sequence number. Each node that hears the request will increase the hash value using *F()*. Nodes can see if the hash value corresponds to the hash chain (and thus to the node), because they know $H_{S \cdot M}$.

When updating the routing tables, nodes follow the normal procedures: update the entry if route is fresher (larger sequence number); if it has the same sequence number, the entry is updated if it has a better (smaller) hop count.

Nodes cannot issue a larger sequence number and decrease the metric, because it is unfeasible to reverse *F()*. Nodes can maintain the hop metric, by not using *F()* on the received value. Colluding nodes can thus advertise better routes than the ones they really have.

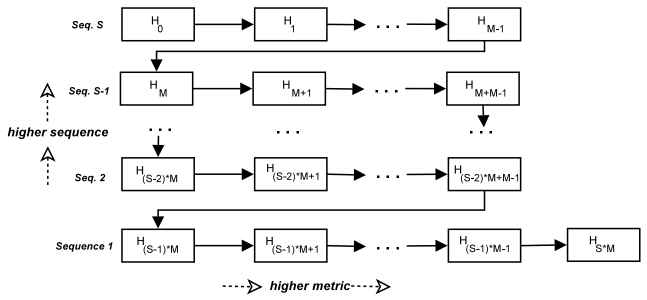

Figure 2: Hash chains for sequence number and hop metric

Trying to force nodes to increase the metric, the authors developed Hash Tree Chains. This construction is based on creating a hash chain where each value is connected to the other through a hash tree. Thus, it is possible to authenticate values and to identify the nodes sending the updates. Using this procedure, intermediary nodes have to increase the metric in order to identify themselves. Receiving nodes check if the node *ID* from which they receive the update matches the *ID* from the hash value received. This construct has some problems in networks with a large number of nodes, since it is possible (with low probability) to overhear hash values that will enable a same distance fraud.

ESMRP also introduces methods for speeding up the verification of hash values, helping in the prevention of DoS attacks, by decreasing the effort needed to verify hash chains.

In conclusion, Ariadne and ESMRP use low complexity constructs to ease the nodes' CPU usage. Regarding trust issues, routing metrics are protected. Encryption is not available as TESLA (keys limited in time) are used as basis. Some protection against replays is possible. Key distribution is not addressed, although shared keys are needed in both.

2.3 Self Securing Ad hoc Wireless Networks

This protocol strays a little from the previous ones, as it does not specifically address routing issues. Nonetheless, it focuses on key problems, which were not tackled by other proposals (although they use them).

The main objectives of SSAWN are to enable encryption, authentication and non-repudiation ubiquity, ensuring high availability of the key system.

Most cryptographic functions are performed in a distributed way.

The following discussion will center on the concepts, leaving the mathematical proofs aside. The interested reader is referred to [11] for further details.

This proposal uses threshold secret sharing (each node has a part of the secret) with the following prerequisites for each node: (1) an unique *ID*, derived from the node's address, that is non forgeable (or forgery is detected by Intrusion Detection System - IDS), (2) a mechanism for local detection of misbehaving nodes (usually an IDS), (3) at least K one-hop neighboring nodes, and (4) a key pair for each node (public and secret keys). As will be seen, the encryption mechanism uses RSA (Rivest Shamir and Adleman) asymmetric keys.

There is a global Secret Key (SK) and the corresponding Public Key (PK). SK is '*divided*' into K parts. The objective is that any K nodes holding a partial secret will form a distributed Certificate Authority (CA)[1] with SK.

Each neighboring node i has a partial secret key that is a function of its *ID* (P_{ID}). The distribution of P_{ID} involves the generation of a polynomial of order K-1, known only in the initial setup. Using Lagrange interpolation, it is possible for K nodes holding a partial secret share to recover SK. However, a coalition of K-1 nodes holding a partial secret share does not have any information about SK.

A node wanting to use the distributed CA must contact K nodes that have a partial secret share (figure 3). These K nodes must be one-hop neighboring nodes. This is due to the fact that it is easier to collect reliable information about misbehavior of closer nodes than multi-hop ones. As is expected PK is known by all nodes.

Each node must have a certificate signed by SK validating its key pair. This certificate has a limited lifetime, to ensure continual renewal. A node must ask K nodes to sign its key pair in order to acquire a valid certificate. The K nodes accept the request if the node has not been convicted of misbehavior, according to each node's internal information. Here IDS systems can be used to gather knowledge about node behavior.

Certificate renewal follows the same principles; however the certificate cannot be in the Certificate

[1]A CA is an entity that issues certificates.

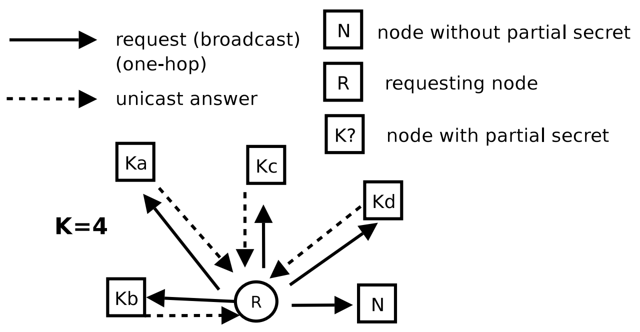

Figure 3: Certificate request in SSAWN

Revocation List (CRL). A node enters this list (which is local to each node) when the list owner, by direct monitoring, observes malicious behavior or when K different signed accusations are received by a node. When a node observes misbehavior it broadcasts its finding, signing the information.

A node can also request a partial secret to K nodes. Using their partial secrets and the requesting node's *ID*, the K nodes issue a partial secret. Each node consults its CRLs as for certificate issuing before answering the node's request.

Partial secret keys are also renewed periodically. Note that the global SK remains the same; what changes are the partial secrets (more precisely the function that generates P_{ID}). Each node (holder of a partial secret) has a probability of starting this function renewal, in which case it uses K nodes to generate an update polynomial. This polynomial is encrypted using SK and broadcasted. Each node that receives the change notification uses K nodes to update its part of the share secret. This works even if the K nodes have not updated their secrets, as long as all K nodes have the same version of the function. Partial secret shares from different versions of the function cannot be used together, which makes impossible the accumulation of partial secret shares through updates by malicious nodes.

Any K nodes can be used to derive the SK. This means that a node can roam to find nodes, collecting results from answering nodes. Therefore, mobility improves the availability of the distributed CA.

Initialization is done by an offline authority that distributes the partial secrets by K initial nodes, or through a coalition of K nodes using collaborative admission control.

Regarding the possibility of malicious nodes sending and/or generating false partial secrets for good nodes, the authors point to Verifiable Secret Sharing (VSS) methods to recover from this problem. This problem and the forgery of nodes IDs could make this proposal vulnerable to *Sybil* attack [14], thus the importance of the IDS system and VSS.

The value of K defines availability (K one-hop nodes need to be reached) and security level (K nodes must be taken over before malicious nodes can became a CA). As is mentioned in the proposal, this can lead to conflicting goals.

The performance evaluation performed by the authors indicates that normal laptops can cope with the computational work; however, low end devices may experience some delays. As is easily perceived, there is some computational work associated with this proposal. Initial deployment, could also pose a problem. Nonetheless, a distributed key certification is achieved, which enables keys distribution. Malicious nodes, if detected by a needed IDS, are ignored by the CA through CRLs.

3 Ad Hoc Secure Routing

In the war against malicious nodes in ad hoc networks we consider that the battlefronts are:

1. **Message integrity** - message information should be protected, MACs should be used;
2. **Signing** - to ensure information origin;
3. **Encryption of information** - the possibility of encryption should exist (in routing and data packets), the user/application could then define its needs;
4. **Key distribution/usage** - this is an important factor, as this must exist to enable striking the second and third fronts;
5. **Computational/Energy savings** - a protocol should try to minimize this spending, because low-end devices are expected to be a large percentage of nodes in these types of networks.

Current proposals do not address the whole picture. However, many of them focus on multiple aspects: ODSBR has signing, message integrity, encryption and path protection; Ariadne/ESMRP deals with easing computational effort, path and metric protection and signing; Self-Securing tackles key distribution/usage. There is no proposal addressing all fronts.

The security protocol presented in this paper, denoted AD hoc SEcure Routing (ADSER), includes: *(2)* - signing (which enables message integrity *(1)*) using as basis ODSBR and Ariadne constructs; *(3)* and *(4)* - availability of keys to allow encryption of data using PK/SK pairs, distributed using a simple algorithm and certified through a distributed CA as described in SSAWN; and finally *(5)* - low complexity tasks, resorting to Ariadne and ESMRP algorithms.

ADSER also defines the protection of path information, resorting to ODSBR and Ariadne for source routing protocols (DSR) and to ESMRP for reactive next hop routing protocols (AODV). The difference between the two protocols resides on the fact that source route uses the overall path and reactive next hop only knows the next hop of the path. Thus, protecting the overall path information is needed in source routing protocols (which also protects the hop count metric), and hop metric protection is needed in reactive next hop routing protocols. This is, of course, based on the premise that hop count is used to rank paths[2]. In both types the sequence number also needs to be secured.

The following protocol will define procedures to enable safe route discovery. This will use hash functions to ease computation, some encryption and signatures to ensure the origin of messages (this will be described in 3.3). The cryptographic functions will use asymmetric public/secret keys. To allow nodes to have the public keys of their corresponding nodes, a way to obtain these keys will be depicted in 3.2. When entering the network a node will have to follow some steps to gather the necessary information to operate in the network; this process will be described in subsection 3.1. The secure of route repairs and maintenance and routing errors is not mentioned in the protocol description, as it follows the same principles of the secure of route request and reply messages.

The detailed description of the protocol will only dwell in the source routing problem (DSR will be used as an example). Source path information will enable a larger level of security, because the overall path is known. Section 4 briefly describes the extensions applied to the security protocol in order to cope with reactive next hop routing protocols (AODV will be used as an example).

Securing data packets is not mentioned, but granting that public/secret key pairs exist, data can be encrypted using these keys.

[2]Other types of metrics would imply different protections.

3.1 Entering the network

When a node is brought to existence in the network (either from boot up or from reaching radio range of the network), it must perform several steps to assure that its PK is distributed to interested nodes. All the procedures described assume that the node's address is already configured.

First, the node will generate an *ID* that will enable SSAWN. As describe in 2.3 this *ID* is derived from the node's address, and shall be unique and not forgeable. Next, the node generates its key pair (public and secret keys). The public key is then certified by the K-coalition of nodes, again using the algorithm of 2.3. A certificate (*CERT*) asserting that pk_{ID} is the public key of node *ID* is the result of this algorithm. The triplet $\{pk_{ID}, ID, CERT\}$ is then broadcasted using a Time To Live (TTL) sufficient for dissemination of the node's PK.

Nodes receiving this broadcast use the K-coalition to verify the certificate. When the certificate holds true, the node caches the triplet, and re-broadcasts it. Certificates have an expiration time; this shall be honored by purging entries from the cache that have expired. Nodes shall also request the global PK for the coalition, to be able to verify signatures and certificates made by the coalition. A global hash function *F()* shall also be received in this setup phase, to enable the hash chains algorithms. This procedure should be correctly protected as malicious node should not be able to trick nodes with false *F()* an PK. This is related to the bootstrap issues of SSAWN.

3.2 Getting Public Keys

When a node needs to encrypt data to other node or verify its signature it needs its PK. A cache of triplets $\{pk_{ID}, ID, CERT\}$ is to be maintained in the node, as mentioned before. Nonetheless, the node may not have the PK of the node with which it wants to correspond. This can be due to the expiration of a previous triplet or its inexistence (broken links, small TTL, node has just moved to a new location, etc). We will describe in algorithm 1 a procedure that will enable nodes to acquire public keys and certificates. This is not based on any of the protocols described so far; it tries to follow common rules of ad hoc protocols.

When a node requires a public key of other node, it broadcasts a Public Key REQuest (PKREQ) (similar RREQ in DSR, described in Appendix B). The contents of the PKREQ messages are presented in algorithm 1. When a node receives the PKREQ, it replies to the request with a Public Key REPly message (PKREP) if it has the public key requested, or it re-broadcasts the PKREQ to find a node with this information. The PKREP message is broadcasted, in contrast with RREP messages in DSR, in order to increase the number of nodes knowing $\{pk_{ID}, ID, CERT\}$.

A node receiving the PKREQ message may ignore it, if it already has received the same request before and it still contains it in cache. To remove the request from the cache there is a TIMEOUT[3] to rate-limit sending new PKREQ. Additionally, a limit is imposed in the number of PKREQ that can be sent[4]. Whenever the TIMEOUT expires, the number of failures increases. In the source node, reaching a defined limit triggers a destination unreachable error to the application. Intermediate nodes do not have application requests. This process serves two purposes. First, it enables re-requests for the same triplet (the request in cache expires and new one is processed). Second, it obviates overwhelming the network resources due to unreachable nodes.

When a node receives a PKREP message, it ignores the message if it already knows the public key being broadcasted, or it stores the triplet in its list if the public key is not known. Also, if it has a PKREQ for that *ID* in cache, it re-broadcasts the PKREP and removes the PKREQ from the cache. This action stems from the fact that nodes should only send replies to queries that they have heard and have not yet answered. Having a cached PKREQ is the confirmation of the existence of an unanswered PKREQ. This also limits PKREPs.

Before caching the triplet the node must verify the certificate (using the globally known PK).

3.3 Route discovery

In this sub-section we will describe the process for route discovery. The primal objective is to protect path information. This protection will be performed based on a combination of ODSBR and Ariadne, to develop a low complexity algorithm. ODSBR allows

[3] In DSR an exponential back-off algorithm is used for RREQs.

[4] This is similar to AODV and its RREQs.

Algorithm 1 Get PK

```
Node needing the PK for node $ID$
  1. Broadcast request for key of the node identified by $ID$.
     The request should carry an $PKREQ_{ID}$ correlated to $ID$
  2. Add $PKREQ_{ID}$ to cache with TIMEOUT
  3. If TIMEOUT expired
       (a) Clear $PKREQ_{ID}$ from cache
       (b) Increase number of failures for $ID$
       (c) If number of failures reaches MAX_FAIL return
           destination unreachable to application and stop
           PKREQ sending
Each node hearing PKREQ
  1. If $PKREQ_{ID}$ in cache ignore
  2. Else If $\{pk_{ID}, ID, CERT\}$ exists, broadcast reply PKREP with TTL=max hops, $PKREP_{ID} = PKREQ_{ID}$ and the $\{pk_{ID}, ID, CERT\}$ (this includes node $ID$)
  3. Else
       (a) Rebroadcast request
       (b) Add $PKREQ_{ID}$ to cache with TIMEOUT
  4. If TIMEOUT expired
       (a) Clear $PKREQ_{ID}$ from cache
       (b) Increase number of failures for $ID$
       (c) If number of failures reaches MAX_FAIL stop
           PKREQ sending
Each node hearing the PKREP
  1. If $\{pk_{ID}, ID, CERT\}$ known ignore
  2. Else if certificate verifies (using the global PK of the coalition)
       (a) Add to local tables
       (b) If $PKREQ_{ID}$ in cache
             i. Rebroadcast PKREP
            ii. Remove $PKREQ_{ID}$from cache
```

the removal of nodes from the end of the list. Ariadne does not protect the removal of every node except the source. The introduction of the hash from Ariadne prevents the first issue (it is unfeasible to reverse $F()$) and the nonce[5] added here thwarts the second. Additionally, using hash functions eases the computational task of nodes when compared to the '*onion*' encryption of ODSBR.

ODSBR uses signature of the nodes as Ariadne, but the latter uses keys with limited time to live (TESLA). Here, these keys are not used as they make temporal restrictions in key verifications and require loosely coupled time synchronization. Shared keys of ODSBR pose a problem of distribution. Therefore, in this case PK/SK pairs (described in the previous sub-section) will be used[6].

[5] Value randomly generated only used once.

[6] Shared keys can be derived after connection establishment using the public keys.

Each node needs to append security extensions to the routing messages in order to secure the routing and the overall path. The security extensions are described in figure 4 for the source node and in figure 6 for intermediate nodes. Figures 5 and 7 illustrate the process.

```
 SSN =  Sign(Option Type|Seq.  Num.|Target
        Addr|Source Addr|nonce_S)
  CS =  Crypt_Dest(SSN|nonce_S|Source Addr)
Hash =  F(SSN|nonce_S|Source Addr)
 SSs =  Sign(Option Type|Seq.  Num.|Target
        Addr|Source Addr|CS)
  SS =  Sign(Option Type|Seq.  Num.|Target
        Addr|Source Addr|CS|Hash)
Mesg =  Normal RREQ|SSs|CS|Hash|SS
```

Where
`Crypt_Dest(Y)` - crypt data Y with public key of destination node
`Sign(Y)` - sign data Y using the private key of the current node
| - denotes data concatenation
`F` - is a hash function
and
`Option Type` - indication of message operation (2 in DSR indicates a RREQ)
`Seq. Num.` - sequence number of request
`nonce_S` - value randomly generated by the source node that can only be used once

Figure 4: Security extensions generated by the source node

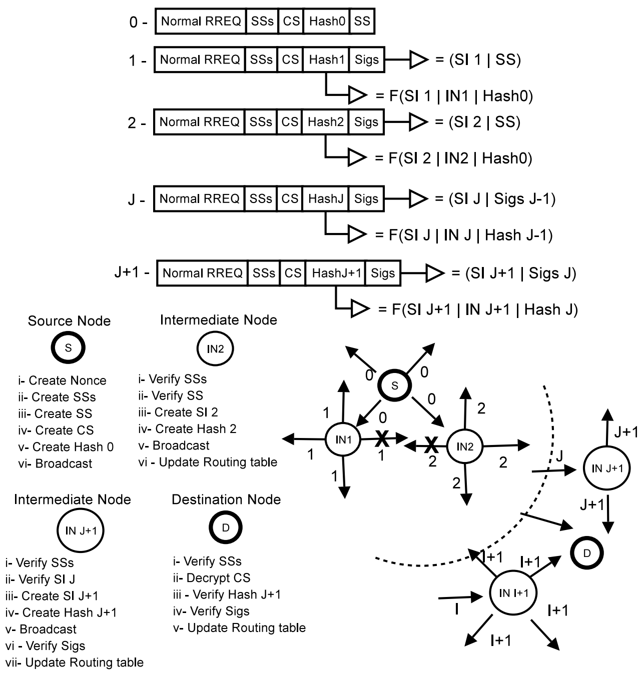

Figure 5: ADSER Routing Request

When a node sends a RREQ, it must be guaranteed that the RREQ was really sent by it. All nodes in the path, including the destination, need to verify that source issued the RREQ. The *SSs* (Signature Source simple) is the signature of all message parts generated by the source, to provide source integrity. The *SSs* generated certifies that the node with *Source Addr* has sent the RREQ (*Option Type* =2) with a specific sequence number (*Seq. Num.*) to search for the destination node *Target Addr.* A Crypted Source (*CS*), explained below, is also included in this signature to provide all information generated by the source to the destination node. The sequence number must also be protected, because malicious nodes could change it to advertise fresher routes using the same signatures. The validation of the source node's signature stops invalid nodes (nodes in CRL) from starting RREQ floods in the network. *SSs* is verified in all nodes.

Two other signatures are required: *SSN* (Signature Source Nonce) and *SS* (Source Signature). *SSN* is built using a nonce to prevent the removal of every node from the node path, leaving only the source node. This is related to the *Hash*, which follows Ariadne, and provides the confirmation of path traversal (described below). Without a nonce, as every node seeing the RREQ has access to *SSs* they could generate *Hash*. In this case *CS* would not exist and thus *SSN* would be *SSs*. *SS*, the other signature, signs the information actually sent on the RREQ to the next hop nodes.

The nonce is encrypted in *CS* with the public key of the destination, so that only this node can extract and verify *SSN* and nonce, and no other node can access this. *SSN* and nonce will serve for the hash chain verification.

The source node then broadcasts the overall message *Mesg* (message 0 in figure 5).

```
Hash =  F(SIs|Node Addr|Hash_previous)
  SI =  Sign(Option Type|Seq.  Num.|Target
        Addr|Source Addr|HC|CS|Node List)
Sigs =  (SI|Sigs_previous)
Mesg =  Normal RREQ|SSs|CS|Hash|Sigs
```

Figure 6: Security extensions generated by intermediate nodes

Intermediate nodes perform similar operations. These nodes generate a *SI* (Signature Intermediate) to certify that they actually processed the message. The receiving node will verify the overall path by validating each signature in the *Sigs* list (which holds all signatures). A *Hash* is generated to be used by the destination node in the hash chain verification. It is calculated from the previous hash, the *SI* and the node address, so to form the required hash chain. A node cannot build a different hash (with only the nodes it wishes) because it has not access to *SSN* and the nonce.

When the request is re-broadcasted, *Mesg* is sent, with the security extensions appended to the RREQ.

Upon receiving a RREQ, an intermediate node verifies the last sender signature (*SI*) and the source node signature (*SSs*), in order to assure the integrity of the source and previous intermediate node. This verification requires access to the PK of the signing nodes. If the triplet is not available the procedure described in 3.2 to get public keys shall be followed. The validation of source node's signature stops invalid nodes (nodes in CRL) from starting RREQ floods in the network. The last sender's signature verification halts immediately invalid nodes from being part of the path.

If the integrity of both nodes is assured, the intermediate node proceeds its normal routing operations. The node verifies each signature in the path, validating the nodes list in the RREQ, before adding this route to its internal table. This check can be done prior to re-broadcasting. That will stop invalid/adulterated RREQ from spreading, but will inflict a delay in the RREQ traversal. Valid paths should occur more often than malicious ones, so overall, sending before verification should prove to be a better option.

When the destination node receives the RREQ it verifies the signature of the source node (*SSs*). Next it decrypts *CS*, and verifies the *Hash*, performing the hash chain verification to ascertain the integrity of the overall path. Only if these checks hold, it will verify the *Sigs* list, validating this list with the one available in the RREQ message. If all verifications hold correct, the node issues a normal RREP and signs the message, including the nodes' list.

The RREP message only needs to contain a signature of the destination (similar to *SS*, but for the destination) and of the intermediate nodes (*SI*). Since we are assuming bidirectional paths, the destination node signs the complete list of nodes between the source and destination (messages 1, 2 and 3 in figure 7).

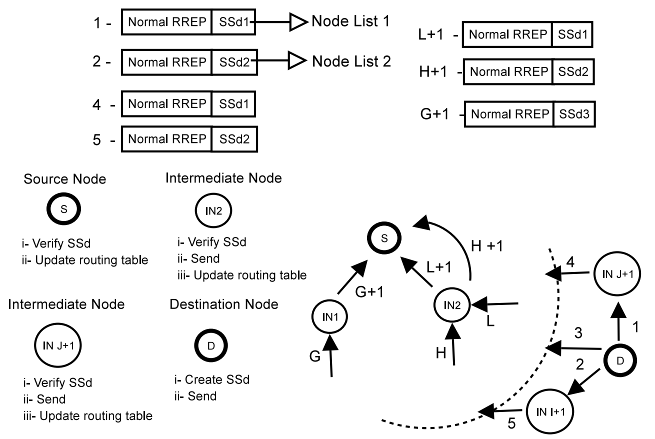

Figure 7: ADSER Routing Reply

The RREP now travels back to the source. Each intermediate node verifies the destination's signature and the one from the previous hop. If this holds true, the RREP is re-broadcasted. The node can now add the path to the destination to its internal routing table. As the nodes' list is signed by the destination, one can trust that the path to the destination is not altered, as it would only harm itself.

When the source node receives the RREP it checks the signatures and adds the nodes' list to its routing table. As in normal DSR, if multiple paths are available, several RREP will arrive, and the source will choose a path using the same algorithms as in DSR (shortest path).

When RREP initiated by intermediate nodes are to be used (see Appendix B), the node can cache the RREP signature and send it when it sees a RREQ. Certificate expiration (and thus key pairs expiration) prevents cache old-age problems for intermediate and source nodes.

3.3.1 Spontaneous RREP

When RREPs initiated by intermediate nodes are to be used (see Appendix B), the node can cache the RREP signature and send it when it sees a RREQ. This signature will however only certify part of the path. As illustrated in figure 8, if *IN2* sends the cached *SSd1* it will certify the previous path {*IN4*, *IN3*, *IN2*, *IN1*}. To use the travelled path *S2* needs confirmation for {*IN4*, *IN3*, *IN5*, *IN6*}. To insure this, the intermediate node will perform as forwarding the RREQ, calculating *Hash*, *SI* and *Sigs*. This will be added in the RREP, as the cached node list and the cached signature. This can be seen in figure 9. The other intermediate nodes (*IN5* and *IN6* in the figure) will be unable to certify the path because they can not verify the *Hash* value. The source node however will, if it has cached the *nonce_S* associated with the RREQ. It will then be able to proceed as the destination when receiving the RREQ, verifying the hash and the *Sigs* list. This will certify the path to the intermediate node that sent the spontaneous reply. The remaining path is certified by the cached signature *SSd* and the cached node list.

The intermediate node should check if the cached signature is still valid, that is, that it has not expired, before sending the RREP. If the signature has expired it must be deleted from the cache and no RREP sent.

Nodes will have to distinguish this reply from the normal one. This can be done using a different *Option Type*.

The destination node will not have a path to the source node. A gratuitous RREP could be sent by node *IN2*, following the RREQ methodology. This would also be a different *Option Type*.

The originating node can nonetheless issue a RREQ increasing the Dst Seq Nr so that Spontaneous RREP will not occur and a complete RREQ, RREP procedure is performed.

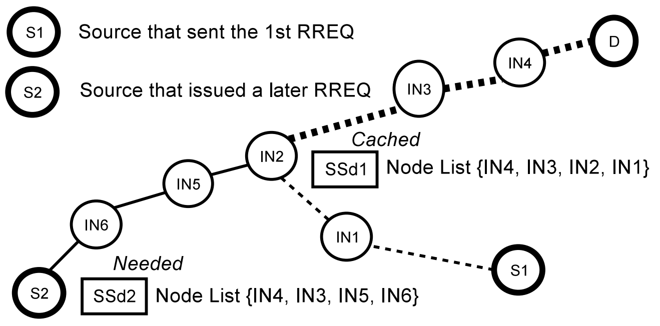
S1
Source that sent the 1st RREQ
S2
Source that issued a later RREQ
D
IN4
IN3
IN2
Cached
SSd1
Node List {IN4, IN3, IN2, IN1}
IN5
IN6
IN1
S1
Needed
S2
SSd2
Node List {IN4, IN3, IN5, IN6}

Figure 8: ADSER Spontaneous Routing Reply

```
Hash =  F(SIs|Node Addr|Hash_previous)
  SI =  Sign(Option Type|Seq.  Num.|Target
        Addr|Source Addr|HC|CS|Node List)
Sigs =  (SI|Sigs_previous)
Mesg =  Normal RREP|Cached SSd|Hash|Sigs|Cached List
```

Figure 9: Security extensions for Spontaneous RREP generated by intermediate nodes

Certificate expiration (and thus key pairs expiration) prevents cache old-age problems for intermediate and source nodes.

4 Extensions to Reactive Next Hop Protocols

The procedure for reactive next hop protocols is similar to the one for source routing. In this sub-section we will briefly describe the relevant differences. We will address the example of AODV routing protocol (described in Appendix B).

As mentioned, when protecting this type of protocol, the hop metric and sequence number will be addressed using the algorithms of ESMRP, that force the nodes to increase the hop metric.

When entering the network, a node will perform the same steps of 3.1. To use ESMRP constructs, each node also needs to build a hash tree chain for each RREQ and RREP it starts.

In RREQ and RREP, nodes will verify the signatures of the requesting node and the sender node. Hop count and sequence number will be validated resorting to the hash tree chain value. The top hash should be sent signed. If correct, this value will be incremented. The message will be signed and sent (broadcasted in RREQ and unicasted in RREP, as per normal routing behavior). At this point, a node can add the information to the routing table if the normal conditions for route updates hold (fresher sequence number, or same sequence number but better hop count).

When the RREQ arrives at the end node, the node performs the same checks. It can also update its routing table and send a signed RREP including the top hash (signed) of a new tree hash chain. This guarantees that in the reverse path verifications can be done to the hop count/sequence number. The requesting node performs the same confirmation procedures and route updates.

The triplets $\{pk_{ID}, ID, CERT\}$ are also used here for signing and validation purposes. Public keys are obtained as described in 3.2. Certificate issuing is also performed using SSAWN.

Reactive next hop protocols, although lighter in routing message exchange, have the disadvantage that the source node does not know which route its packets will take before reaching their destinations. Although some efforts can be made to secure this traversal, source routing provides more knowledge about packets' intended voyage.

5 Evaluation

In this sub-section we will address several global aspects of the protocol described. It will serve as a wrap-up and to pinpoint some issues.

The protocol described, ADSER, tries to secure a route discovery process and to distribute keys to all nodes, to allow encryption and signature, using low complexity functions when possible.

As mentioned, techniques from ODSBR, Ariadne/ESMRP and SSAWN were used to achieve this. Additions were made to the route discovery process to mitigate node suppression.

The portrayed protocol addresses/mitigates the following issues:

- **Eavesdropping** - the deployment of PK/SK pairs and their associated signature allow nodes to encrypt data and prevent this issue;
- **Identity problems** - the signatures used allow the verification of identity. Together with CRL, nodes can be prevented from operating in the network. This authorization should be more fine grained (a node should be authorized for some things but not others). The population of CRL should also be more defined. The problem of certificate stealing is not addressed. To achieve this, a node should assume the *ID* of the intended node and steal its certificate. The *ID* theft should be protected by IDS;
- **Trust issues** - care was taken in validating relevant information regarding routing protocols. Nodes are thus prevented from advertising better route characteristics than what they have and/or lie about that information. Selfishness was not addressed here. Although this can be an issue to take into account, it is more related to fairness than security issues;
- **Replays** - by using sequence number and nonces, ADSER deals with the replay of previous messages. Sequence number and nonce generation was not portrayed, but should be a protected procedure.

DoS attacks were not discussed. Although some control can be made through the use of CRL lists to ignore nodes, this was not exploited to full potential in the text. This brings to focus the use of IDSs in the network. They should not grow to be a full blown IDS system, but rather the sufficient to evaluate node behavior (in accordance to some defined factors). The use of an IDS could also provide information regarding route metrics. ODSBR uses a link failure detection approach that, in our opinion, demands too much message exchange without the proportional benefits. However, a security metric should be obtained so that nodes can choose a path based on its security qualities.

The protocol discussed here tried not to abuse network resources (bandwidth and nodes' CPU), but the use of the coalition of K nodes in SSAWN, involves a bit of both resources. However, they will only be used when a node needs to sign its triplet (and in partial secrets update). Certification expiration time will be a factor to balance between security and resources usage (together with K as discussed in 2.3).

6 Conclusions

In this paper, a security protocol for ad hoc networks, ADSER, was presented. This protocol includes mechanisms to cope with the majority of the security issues of ad hoc networks, and tries to secure a route discovery process and to distribute keys to all nodes, to allow encryption and signature making, using low complexity functions when possible. ADSER takes as a baseline some current security protocols and addresses the secure routing concerns of both source routing and reactive next hop protocols. This protocol is able to mitigate eavesdropping through the encryption of data, identity problems through signatures, trust issues through prevention of wrong route advertisements, and replays through sequence number and unique values generation.

As future work, it is planned to study the interaction of ADSER with IDS, and to address security issues of bootstrap phase. Our plan is also to address, through network simulations, the overhead due to the security process and the performance evaluation of ADSER.

A Hash Chains

Hash chains are based on one way functions. These functions cannot be reversed, so if we compute $h = F(j)$, being *F()* a one way function or hash, there is no feasible computational way of deriving j from h. This means that if we release h as result of a hash then we must also have/know j. Hash chains are built applying the one way function recursively, that is $H_k = F(H_{k-1})$, where each H_k for $0 < k < N$ is an element of the chain. H_N is the top of the chain, or top hash. A value J is part of the chain if it is possible to hash it to get H_N, that is $H_N = F^{\mathrm{k}}(J)$, meaning that $J = H_j$ and $k = N$-j, for some $0 < j < N$.

B AODV and DSR

The objective of these protocols is to find routes in an ad hoc network. Both are on-demand, meaning that only when a node wants to transmit data does it search for a route to the destination. AODV (Ad hoc On demand Distance Vector) [2] has reached a RFC form and DSR (Dynamic Source Routing) [1] is currently an Internet Draft from IETF. DSR is source based (each packet has the route to be traveled explicitly set), whereas AODV is table driven (each node that receives the packet has to perform a routing table lookup to discover the next hop for the given destination). There are other ad-hoc routing protocols that also use tables lookup, but differ from AODV. This leads to the use of 'reactive next hop' routing protocol to characterize it (reactive in the sense of on-demand).

The protocols, however, share some common principles of operation. When a node needs to reach another one, it consults its internal cache for the route. If it does not find one, it broadcasts a RREQ (Route REQuest) to query the network. Nodes hearing the broadcast should make the same check on their internal cache. If they have a route for the destination node or if they are the destination node, they should originate a RREP (Route REPly). If none of the previous conditions hold, the node must re-broadcast the RREQ, but only if it is the first time it sees the request (request identifier in the RREQ allows this verification). As these requests are broadcast, each node (including the destination) will hear the same request repeated, arriving by different paths.

The RREP differs in each protocol. In DSR all RREQs are replied by the destination, originating multiple RREPs. This behavior of DSR enables caching different routes to the same destination. In AODV only one route is known for each destination and a destination node only sends one RREP. That means that, a RREP is only sent for the first RREQ received.

In DSR the RREQ and RREP messages have a field that contains all nodes where the packet has passed, which allows the node issuing the RREP to use this field to address the RREP. The node that started the request (and that receives the various RREP) has its answer from this field. Nodes hearing the RREQ or RREP can use the path field to update their internal routing table. In AODV, only the next hop for a given destination is known. Thus, when forwarding RREP or RREQ, nodes update the routing table to the destination or source respectively with the node that sent the message. The 'construction' of this path is the answer that the route requester receives.

References

[1] Y.-C. H. David B. Johnson, David A. Maltz, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR), Internet Draft,draft-ietf-manet-dsr-10.txt," April 2003.

[2] S. D. C. Perkins, E. Belding-Royer, "Ad hoc On-Demand Distance Vector (AODV) Routing," Internet Engineering Task Force, RFC 3561, July 2003.

[3] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Mobile Computing and Networking*, 2000, pp. 255–265.

[4] J. Y. L. B. S. Buchegger, "Performance Analysis of the CONFIDANT Protocol (Cooperation Of Nodes - Fairness In Dynamic Ad-hoc NeTworks)," in *Mobile Internet Workshop*, 2002.

[5] S. Yi, P. Naldurg, and R. Kravets, "Security-aware ad hoc routing for wireless networks," in *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC 2001)*, Oct 2001.

[6] M. G. Zapata, "Secure Ad hoc On-Demand Distance Vector (SAODV) Routing, Internet Draft,draft–guerrero-manet-saodv-00.txt," October 2001.

[7] P. Papadimitratos and Z. J. Haas, "Secure routing for mobile ad hoc networks," in *Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, January 2002.

[8] C. N.-R. Baruch Awerbuch, David Holmer and H. Rubens, "An On-Demand Secure Routing Protocol Resilient to Byzantine Failures," *ACM Workshop on Wireless Security (WiSe)*, September 2002.

[9] Y.-C. Hu, "Ariadne:: A secure on-demand routing protocol for ad hoc networks," in *Proceedings of the 8th annual international conference on Mobile computing and networking.* ACM Press, 2002, pp. 12–23.

[10] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Efficient security mechanisms for routing protocols," in *Network and Distributed System Security Symposium, NDSS '03*, February 2003.

[11] H. Luo, P. Zefros, J. Kong, S. Lu, and L. Zhang, "Self-securing ad hoc wireless networks," in *Seventh IEEE Symposium on Computers and Communications (ISCC '02)*, 2002.

[12] A. Shamir, "How to share a secret," *Communications ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[13] D. S. A. Perrig, R. Canetti and D. Tygar, "Efficient and secure source authentication for multicast," in *Network and Distributed System Security Symposium*, 2001.

[14] J. Douceur, "The sybil attack," *First International Workshop on Peer-to-Peer Systems, (IPTPS '02)*, March 2002.