Ad-Hoc Routing Security Report

Pedro Brandão Susana Sargento
Sérgio Crisóstomo Rui Prior

Technical Report Series: DCC-2003-7

Departamento de Ciência de Computadores – Faculdade de Ciências
&
Laboratório de Inteligência Artificial e Ciência de Computadores

Universidade do Porto
Rua do Campo Alegre, 823 4150 Porto, Portugal
Tel: +351+2+6078830 – Fax: +351+2+6003654
http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

1 * Intro *

The purpose of this report is to analyze some current routing solutions to security problems in ad-hoc networks. We will summarly describe the proposals, and compare them, emphasizing the main goals of each. The shortcommings/problems will also be noted.
We will start by, in the first section, introduce the characteristics and security problems of ad-hoc networks. We will then describe some protocols that try to cope with these problems.
We will end the report with some comments and conclusions regarding this analysis.

2 * Contents *

3 * Context *

Ad hoc networks are a new emerging technology (the term emerging may however be an overstatement). Their main objective is to enable the autonomous creation of communication channels between devices. These channels should adapt to highly dynamical network configurations, with nodes entering and leaving the network and no knowledge of what nodes are connected prior to entering the network. The configuration of the node is to be done with the least assumptions possible: the number of nodes that exist is not known, there are no particular node types (no central servers expected), user interaction to configure the connection should be minimal (if at all).

When the number of nodes increases and communications cannot be done point to point (either due to a range limit on wireless links or non-direct connection in wired links), there is the need of cooperation to forward packets to their destiny. This required cooperation is the source of the majority of security problems associated with ad-hoc. Nodes that are asked to forward packets are not inherently good nodes, and knowledge of *goodness* is difficult in a network where nodes are not known before their entrance in the network.

Another security issue is the network layer used. Although, wired links may be employed (in which the following is less of a problem), the vast majority (if not all) of the communication channels used will be radio waves. In this medium, as one can easily perceive, it is very simple to listen to other nodes communications

(regarding physical access to the medium). All that it is needed is a wireless card and to be in the range of the transmission.

Adding to these disadvantages, most of the nodes in these type of networks are small powered, in processing and battery capacity.

These characeristics lead to the following security concerns:

+ Easy eavesdropping
  - as said it it is simple to *'hear'* the data transit that literally flows in the air.
+ Denial of Service (DoS)
  - there are several variants of these attacks:
    - > packets flood - the usual DoS attack: flood the victim with more requests than it can handle. This is the general objective, accomplished with the more specific attacks detailed below.
    - > radio jamming - as the physical medium is shared by all nodes in range, it is possible to flood the medium with bogus data, invalidating any other communication. This is however out of the scope of this report (and the protocols studied). There are techniques associated with the physical medium to surpass this; examples of this use frequency hoping spread spectrum techniques.
    - > battery exhaustion - as mentioned, we are dealing with small powered devices. If one sends a flood of packets to be forwarded by a victim node, its battery could be exhaust just serving others needs.
    - > CPU exhaustion - when defining security measures, one should be aware of the complexity of the algorithms used. It may be possible for an attacker to send bogus packets, which will oblige the victims' CPU to make complex tasks to discern them as bogus. This allows a DoS attack.

  All these attacks could also be done in a distributed manner, thus scalating to a Distributed DoS (DDoS). In this report, however, the concern will be mitigating DoS (which does not automatically imply mitigating DDoS).
+ Identity problems
  - these problems are less specific of this type of network but their resolution is highly dependant of the architecture/protocols used. One must deal with:
    - > (un)authorization - nodes should only be allowed to execute actions that they are authorized to do. This implies a mechanism to associate actions to levels of authorization and to enforce the authorization verification.
    - > impersonation - nodes can use eavesdropping or capture/theft to acquire information that will enable them to assume the identity of other node. One has to guarantee that nodes are who they pretend to be. The confirmation routine should not allow other nodes (that were listening) to assume/steal the identity of the node. If a node is captured through software/hardware flaws and/or misconfigurations or if it is physically stolen, the information it has (keys, passwords, code authorizations, etc) should not permit impersonation or information gathering in order to impersonate or escalate privileges.
+ Trust issues
  - the trust discussed at the beginning of the section associated with the previous point make the following deception attacks possible:
    - > black-hole and/or delays - nodes can wrongfully advertise good forwarding characteristics (or use information poisoning (see below)), to make nodes route packets through them. These packets may not be delivered or delayed in favor of other packets (that the attacker has interest in delivering first). This also increases the scope of eavesdropping as packets forwarded to the malicious node may come from more distant nodes, thus enabling snooping to broaden from packets of neighbour nodes (ones in reception range) to packets that are routed from afar to the malicious node.
    - > information poisoning - there is the possibility of malicious nodes sending wrong routing information, that will be benefic to them (routes will travel through them or other colluding nodes, routing loops can be induced, thus causing DoS). This is similar to the previous point, however it is not restricted to own characteristics advertisement, but also encompass the modification of routing messages from other nodes. The objective is not constrained to attract packet routes but could also be to wreak havoc the routing infrastructure.
    - > wormhole - in this attack two colluding nodes have a fast/direct connection between them, that surpasses the non-malicious slower/multi-hop connection. This way the nodes can (truthfully)

advertise good routing characteristics. The existence of the connection is not the problem, the intentions of the nodes using this link are the security issue.

- > selfishness - although not evilly intended, nodes can decide that they will only forward packets that have interest to them. Their primary objective being to save CPU and battery. This is not really a security issue and there is no identity problem here. This will however prejudice good behaving nodes in a way similar (but in a completely different scale) to a DoS attack.

+ Replays

- in these attacks malicious nodes retransmit packets sent previously (identification/authorization packets, routing packets that advertise routes that are now invalid, etc). These attacks share a resemblance with identity problems, as the source of the information is wrongfully identified (if the replay attack works).

The level of security demanded can vary, and determine certain restrictions to the security architecture. In a military ad-hoc network (a common example in this type of literature is a task force deployed in a hostil environment that needs network communications), there is a mandatory requisite to exclude non identified nodes from the network, and eavesdropping on the network is not even a possibility. Even network existance must be concealed.
However, in civilian networks, one can even question whether to remove a bad node from our usable nodes, or to keep them so to contact a given node that would be unreachable otherwise.
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.

He have identified the primary security issues that are of concern in ad-hoc networks. The protocols that will be discussed try to address some of these problems. We will discussed them based on the points defined.

4 * Basic Definitions *

4.1 # Hash Chains

Hash chains are based on one way functions. These functions cannot be reversed, so if we calculate h = F(j), being F a one way function or hash, there is no computational feasible way of deriving j from h without knowing F. 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 $H_N = F^k(J)$, meaning that $J = H_j$ and $k = N-j$, for some $0 < j < N$.

4.2 # Keys, Signatures and Certificates

Keys are used to encrypt data and to sign content. There are two generic types of key uses: symmetric and asymmetric.

In symmetric keys each intervenient in the process shares the same key. If node A wants to encrypt something to node B it uses Secret Key (S_K) to encrypt the data. Node B must also have S_K, so it can decrypt the message. The messages could be sent to any node that has S_K, as it would be able to decrypt the information. The key would have to be distributed to intended nodes in a secure way, because its disclosure would mean that the encrypted data was no longer secure.
Each two nodes wanting to establish a secure communication must share a different key, so that no other node can eavesdrop their messages (if node A wants to establish a communication with B and another with C, it must have a different shared key with each, which means two different keys).

With asymmetric keys, each node has a pair of keys, a public key (P_K) and a secret one (S_K). As for its name, P_{Ki} is known by every node wishing to communicate with node i. S_{Ki} must be secret and only known by node i. To send data to node i, one uses P_{Ki} to encrypt the data. Only with S_{Ki} can this message be decrypted. S_{Ki} does not need to be distributed, as only node i uses it. To send encrypted data to multiple nodes, one as to encrypt it

with each recipients public key. But, for each node, only one key pair is needed, independently of the number of correspondent nodes.

To ensure that a message was really sent by the specified source, data should be signed. In symmetric and asymmetric encryption, the sender uses S_K to sign the data. The receiver can then check the signature using S_K (symmetric encryption) or P_{Ki} (asymmetric encryption).

Certificates are a way to ascertain the credibility of data. A certificate testifies that presented data is truthful. It can be used to certificate that a P_{Ki} is a valid public key of node i. This way a node can distribute its P_K and nodes that receive it can check if it is valid.
The certificate creation and verification is done using a Certificate Authority (the creation part is usually done through a Registration Authority (RA), but it is simpler to suppose only one entity). This entity (a machine in a network, a human that imprints certificates, etc) issues certificates (this step should be done after checking the veracity of the information being certificated), signing the P_{Ki} with its (the CA's) secret key. Node i can then present its P_{Ki} with the certificate. Nodes check the information using the P_K of the CA to verify the certificate. This P_K (from the CA) must be accessible through trusted and secure means (already present in the node, in trusted servers, etc). The CA's P_K is self certificated and ultimately trusted.
The certificate contains an identification of the owner of the data, that bonds the data to its owner. When the certified P_{Ki} is presented, the receiver nodes know (from the certificate) the identity of the owner of P_{Ki}.

4.3 # AODV and DSR basics

As will be seen, most protocols discussed here use DSR or AODV principles to operate. We will introduce here their basic foundations.

Both these protocols are on-demand, meaning that only when a node wants to transmit data does it search for a route to the destination. AODV has reached a RFC form and DSR is currently an Internet Draft from IETF.
The major difference between the two protocols is that DSR is source based (each packet has the route to be travelled (every hop) explicitly set) and AODV only has the address of the destination node (each node that receives the packet has to do a routing table lockup to discover the next hop for the given destination).

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 routing request (RREQ) 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 route reply (RREP). If none of the previous conditions hold, the node must rebroadcast the RREQ, but only if it is the first time it sees the request; to check this, a request identifier in the RREQ is used.

The RREP differs in each protocol.
AODV only sends a RREP per RREQ. That means that, a node only sends a RREP for the first RREQ it gets (identified by destination and the request identifier). In DSR all RREQs are replied, originating multiple RREP. This behaviour of DSR enables each node to cache different routes to the same destination. In AODV only one route is known for each destination.
In DSR the RREQ and RREP have a field that contains all nodes where the packet has passed. This way the node issuing the RREP can use the path from this field to send the RREP (although it could use another route). The node that issued the RREQ (and that receives the RREP) uses this field to know the route to the destination. Nodes hearing the RREQ or RREP can use the path field to update their internal routing table.
In AODV nodes update their routing tables, through RREP and RREQ. In RREP, the route for the destination node is updated to the node that sent the RREP if the information in the RREP is newer or if it is as new as the node's current one and the hop count is smaller. In RREQ, node's update the routing tables to the source node using the same principles.

Both protocols use routing error messages (RERR) to report error in paths.
In AODV each node has a table with nodes in their neighboord, reporting which nodes use others to access a destination. If a node is unable to send a packet to a next hop neighboor, it sends an RERR to each of its

neighboor that (in its local table) uses that, now faulty, node to reach someone.
DSR only sends RERR to the reverse path of the packet's source route.

Table entries in AODV expire if not used. They can also be removed in case of a RERR.
In DSR entries do not expire and are only removed due to RERR.

In the following table we present a summary of the protocolos.

AODV	DSR
On-demand protocol	
Use of RREQ to discover route to destination node	
RREP determines the route	
Intermediary nodes can send RREP if path is known from internal cache	
RERR remove routing entries	
Routing table with next hop for destination	Several complete routes for destination in routing cache
Only next hop is known	Complete path is known (source routing)
RREP carries number of hops travelled	RREP and RREQ carry path travelled
Intermediary nodes update routing information from RREP and RREQ that travel through them	RREP and RREQ listened (even promiscuously) are used to update routing information
Table entries expire if not used	No expiration of route cache entries
RERR are sent to node that used the faulty node	RERR are sent to reverse path of packet that triggered the detection of faulty node
When a path gets RERR, a new RREQ must be issued	More errors due to slow propagation of RERR and rapid propagation of (stale) route information

5 * Protos Analysis *

We will discuss several proposals to increase security in ad-hoc routing protocols. Some (namely the first two addressed) are more concerned with ensuring/enforcing good behaviour from the nodes in the network. They emphasize on *node watching* to find good routes.
The third proposal also tries to detect links fault locations, but using a different reactive (as opposed to proactive) approach. It also adds cryptographic schemes to guarantee data integrity, confidentiality and signing.
The fourth proposal is an extension (in draft form) to AODV. Therefore it uses its message extensions to add integrity and signing to the IETF protocol.
The next protocol will also use AODV as basis, but it can extend any on demand protocol. It uses shared

symmetric keys to define security levels and according security paths.
The sixth and seventh proposals are from the same authors and aim at lowering computational effort by using one way functions and hash chains to secure respectively DSR and distance vector protocols respectively.
The last discussed protocol is the only one that specifically addresses the key distribution problem. It proposes the definition of a distributed CA using threshold secret sharing.

5.1 # Mitigating Routing Misbehavior in Mobile Ad Hoc Networks Sergio Marti, T.J. Giuli, Kevin Lai and Mary Baker 2000

This proposal tries to provide secure paths to the packets route.

Easy eavesdropping	Not addressed
Denial of Service	Not addressed
Identity problems	Not addressed
Trust issues	If malicious paths are correctly identified these problems can be obviated
Replays	Not addressed

This protocol extends DSR, adding two components, the watchdog and the Path Rater. As it enforces a specific path it uses a source routing protocol. The basic idea is to add a new metric for choosing paths, which measures the behaviour of nodes.

One of the components is the *'watchdog'*, which is responsible for detecting nodes misbehavior . It uses the possibility of listening promiscuously to the medium, to check if the node it forwarded the packet to has transmited the packet accordingly. As it knows (using the source route protocol) the node where the packet should be in two hops, it can also check the correctness of the addressee.
The packets sent are kept in memory until they are seen in the medium or a timeout occurs. If the timeout occurs or there was something wrong in the packet seen, the watchdog judges that the node is misbehaving and sends a message to the source node of the packet. Packets' modifications can also be detected. This mechanism incurs in high memory and processor capacity usage to: store packets sent (longer than normal), listen to communications promiscuously and compare packets sent to packets listened. The content comparison precludes the use of encryption hop by hop. The watchdog has the following weaknesses that are stated in the proposal paper:

- collisions issues - when colisions occur there are no guarantees of the results. If the collision is in the listening node's range, it cannot detect packet forwarding of a previously sent packet; even if the listening node hears its packet being forwarded there could, nonetheless, have occured a collision in the next hop.
- nodes can control transmission power deceiving the watcher node (transmission power is set to a level high enough to permit the watcher's detection, but low enough to not reach the intended receiver)
- colluding nodes can elude the watchdog

The *Path Rater* is the second component which measures the quality of a path. The algorithm uses an average of the nodes' rating to evaluate the path. This rating is compiled from link breaks, active nodes (where a packet was successfully sent in a previous time interval) and watchdog accusations.
Watchdog accusations count as a highly negative input to the metric, and implies the exclusion of this path as viable.

Node accusations (that lead to path exclusions) are not reversible, that is, although there is a mention of a node recovering from an accusation (that can be a wrong one due to the mentioned problems) there is no clear/secure method to implement this (a timeout to increase the node metric or bring it to a positive value is suggested).
The trust in the reports of other nodes' watchdogs is not defined, although mentioned as future work. The use of these reports is not clear, although one is lead to assume that the Path Rater uses them as accusations made by the node's own watchdog (perhaps using a different weight).
Another issue not addressed (and brought up in the next paper) is the lack of fairness, in that nodes that do drop packets are *rewarded* by not receiving packets to forward. These bad nodes are avoided in routes by the path

rater. Their packets however, are routed as usual.

Advantages	Disadvantages
• Avoid bad routes • Detection of packet modification • Detection of routing misbehavior • Trust (except wormholes) issues mentioned are dealt with, provided we can rate nodes correctly (selfishness is detected but not penalized)	• High processing/storage in nodes • The problem of incorrect accusations is only mentioned, but not treated • Content comparison inhibits the use of encryption hop-by-hop • Colluding nodes can elude the watchdog • The watchdog has some possible weaknesses • Inherent trust on other nodes reports, leading to false accusations

5.2 # Performance Analysis of the CONFIDANT Protocol - Cooperation Of Nodes: Fairness In Distributed Ad-hoc NeTworks Sonja Buchegger, Jean-Yves Le Boudec 2002

This protocol's primary objective is to deal with misbehaving/selfish nodes. It shares some characteristics with the previous proposal.

Easy eavesdropping	Not addressed
Denial of Service	Nodes in black lists are ignored
Identity problems	Not addressed, although it mentions imprinting friendship relations and weighted trust
Trust issues	If malicious nodes are correctly identified these problems can be obviated. However, the identification issue is only scratched
Replays	Not addressed

CONFIDANT is based on DSR, as it needs to manage the path traversed by packets. It aims at thwarting the advertisement of false routing information and the tampering with routing protocol headers.

One of its essencial components is the *Neighborhood watch*. Nodes watch over each other to see if packets are forwarded correctly. It is based on the same premise of a shared environment as before, enabling detection of content change or packet dropping, which leads to the same high processing in nodes of the previous protocol. As stated earlier, content comparison does not allow encryption hop by hop. In this protocol, however, content comparison is considered secondary.
Although DSR is used, the requirement is to be based on a reactive source routing protocol in order to know the two hop destination of the packet and thus ensure that the forward packet is correctly addressed.

The misbehavior is reported through ALARM messages sent unicast to interested nodes (the source node and friend nodes). The trust in these messages is treated using *imprinted friendship*. Nodes have (by means not discussed) knowledge of the signatures of nodes that are their friends. This enables a weighted trust in reports that are signed by several nodes (the presence of known friends increases the trustworthiness).
This then leads to a reputation system where nodes move in the ranking according to the information gathered. Own information weights more than information reported by other nodes (where friend nodes are more trusted).

Nodes can be black listed, and in this way ignored by the node that black listed them. A scheme for re-socialization (or reintegration) of the node is mentioned, but concretely only with black lists timeouts can a node repend.

The *Path Manager* uses the reputation system to delete routes that use intolerable nodes. This shows the need for source routing, as the node that sends the packet is the one that chooses the route to use. Paths are ranked acording to the security metric. This component can also alert nodes when requests arrive with malicious nodes in the path.

As can be seen CONFIDANT is very similar to the previous protocol. The main difference is that a black list is used to ignore nodes, so not to receive their packets. The previous proposal only used the information gathered to exclude nodes from paths to the destination, not caring about their source.
Another distinction is the use of trustiness relationships to rate reports. CONFIDANT has a more defined notion of what to do with other nodes reports. Mitigating left as future work the use of other nodes watchdog assesments.

Both use promiscuous listening to compare packets sent to packet listened. Both suffer from the problems stated: collision issues, transmission power control and colluding nodes. The processing in nodes is high in both proposals, motivated by packets storage, listening to communication not addressed to the node and comparison of packets.

Advantages	Disadvantages
• Nodes that do not forward are punished by being ignored • DoS can be prevented if one can identify the malicious nodes • Avoid possible bad routes • Detection of packet modification • Detection of routing misbehavior • The trust based problems mentioned can be tackled, provided nodes are rated correctly	• High processing/storage in nodes • Reintegration of repentant nodes is not addressed (except by timeouts) • Content comparison precludes the use of encryption hop-by-hop • Friend making is not well established • Nodes are unable to identify other nodes (except for the friendship relation) • The *watch* has some possible weaknesses

5.3 # An On-Demand Secure Routing Protocol Resilient to Byzantine Failures
Baruch Awerbuch, David Holmer, Cristina Nita-Rotaru and Herbert Rubens September 28 2002 ↑

The aim of this proposal is to introduce a routing algorithm that is able to cope with byzantine failures. For this, data is encrypted and signed, a method for detecting faulty links is portrayed, and route discovery is based on a metric that weights the faultiness of the links.

Easy eavesdropping	Data is encrypted between sender and receiver
Denial of Service	Not addressed
Identity problems	Packets are signed, unauthorized nodes can thus be detected
Trust issues	Faulty links (black holes, selfishness) are detected and avoided HMACs are calculated for every packet
Replays	Sequence numbers are used, but not mentioned to be part of replay avoidance

This protocol stems from the principle that only source and destination are to be trusted. Data in data packets is encrypted with a shared key between the source and the 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 made using shared keys.
Route discovery follows the principles of on demand protocols: the source broadcasts a Route Request and waits for a Route Response. The Route Request packet is signed and carries a sequence number. Each node checks that the packet comes from an authorized node (checking the signature) and then signs and forwards the flood if it did not see this request before (as in normal on demand protocols).
When the destination node receives the request, it generates a signed response (including the sequence number) with an empty node list. This list is to be populated by each node in the return path. Therefore, when a node receives a response packet it calculates the total cost so far, using its internal weight list and the nodes list from the packet. If the total computed cost is lower than the previous one (or if this is the first response seen from source and destination node, with that sequence number), the node checks the signature of all nodes (verifying the path travelled), adds itself to the path, signs the packet and broadcasts it. If the cost is higher or equal, or if the signatures are not correct the packet is dropped.
The signature check should probably be done before the cost calculation. As it is, a malicious node can alter the nodes list (adding or changing nodes) so that this path would be a heavier one. For this to work a previous path (favorable to the malicious node) must have been already seen. The advantage of the current approach is resource savings, as the node only computes signatures if the cost is less than the previous smaller cost.
When the source node receives the packet it uses the same algorithm (except broadcasting the packet), and updates its path list accordingly. The source node can then use the lighter path to the destination.

Fault detection is done using acknowledgements from data packets. The procedure is as follows: the source node encrypts the data for the destination node and adds a packet counter (which identifies the packet) to the information to be sent. The destination node should send an acknowledgement for each data packet received (using the counter to identify them). A treshold for non-acknowledged packets exists. When the number of missed acks exceeds this threshold the fault detection mechanism is triggered. The data packets used for probing are from new data (we deduced this, altough it is not clear if data is resent or if probing is done using normal flow data).
The mechanism probes the path to identify the faulty link. To accomplish this it requests acks from nodes in the path to the destination. This is achieved by adding the node's id in a probing list that exists in the data packets. The objective is to do a binary search on the path.
After the first failure (the first time the treshold is exceeded) the node in the middle is added to the probing list. If it answers (the source node receives its ack) we infer that the faulty link is somewhere between the middle node an the destination node. If a second failure is detected (the treshold is again exceeded), we deduce that the faulty link is between the source and the middle node. The node in the middle of the presumed faulty path is added to the probing list and the algorithm proceeds.
Nodes only acknowledges packets that have their id in the probing list. As can be perceived, nodes are added to the list, so if the fault on a links was temporary, the destination node will send an ack for the packet.
As can be seen in the figure node 4 is requested to send an ack after the first failure and 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, meaning that 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. Additionaly, each layer also has a HMAC of the destination, source, and encrypted data. The HMAC is generated using the shared key of the node to which this layer is meant. This enables each probed node to check if data was not altered.

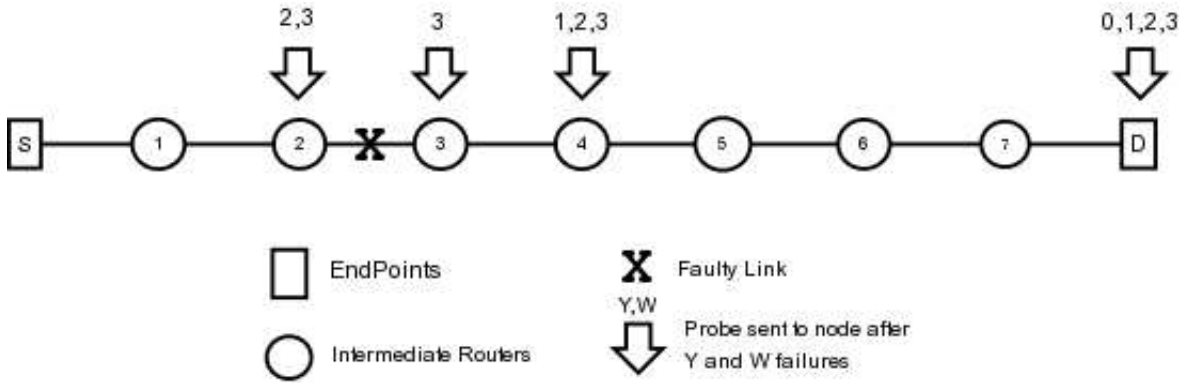


Fig 1 - Fault Detection on An On-Demand Secure Routing Protocol Resilient to Byzantine Failures

Each probed node waits a specified time before sending its ack, in order to combine acks from the following nodes. The source node thus expects to receive the acks combined in a specified order. This eliminates the possibility of a malicious node arbitrarly dropping ack packets, as to incriminate specific links. The acks are again *'onion'* encrypted.
This binary search will enable detection of the faulty link with log(n) probes, where n is the number of nodes. As it is mentioned in the proposal, this will only detect an error between two nodes; if node colluding involves more nodes this process will fail to determine the correct faulty link.
For speeding up the encryption and signing processes, shared keys are used between each pair of nodes. The authors state that a key exchange should occur integrated with the fault detection. This integration is however out of the scope of the proposal. .

The fault detection serves as input for the weight list. This means that a heavy weight links is more unreliable than a lighter one. A link can lighten is weight if ack packets start comming through.
Wormholes attacks are not mitigated by this protocol, as is mentioned in the paper.

The main purpose of this first three proposals is similar, in that they try to discover badly behaved nodes in the network. The first two 'watched' the one hop neighbours, this one tries to detect the faulty link in the travelled path.
The work done by each node is lightened, as nodes only have to react to faults (and not listen to each packet sent). Data packets are sent as normal traffic with extra headers for signing/encrypting and fault detection. The cryptographic operations allied with fault detection (which implies an extra use of these operations) will undoubtedly lead to a processing penalty in nodes. These cryptographic operations are however an advantage as encryption and signatures are used to secure data packets; in the previous proposals no data encryption, hashing and/or signing was addressed.
This proposal also protects the routing information, securing the information received by the node that issued a RREQ.

Advantages	Disadvantages
• Signing of packets • Encryption of data • HMAC of packets • Black Hole and Information poisoning mitigated	• In the route discovery Route Response, the signature check should probably be done before the cost calculation • Possible processing penalties in nodes • Key distribution is not addressed • No DoS defense

5.4 # Secure Ad hoc On-Demand Distance Vector (SAODV) Routing Manel Guerrero Zapata 10 October 2001 ↑

This proposal is an extension to the AODV[AODV] routing protocol from IETF. Aimed at securing the routing protocol, its main objectives, as stated in the paper, are that a malicious node can only cause damage to the routing protocol by not replying to certain routing messages and to lie with information about itself. Which, in either case, the protocol deals as non existing nodes.

Easy eavesdropping	Not addressed
Denial of Service	Not addressed
Identity problems	Nodes use digital signatures or public/private keys for authentication
Trust issues	Hashes are used as a means to validate metric counts
Replays	Sequence numbers are used, so to mitigate replay attacks

The proposal uses new extension messages for AODV. It relies on hash chains and digital signatures. The distribution of these signatures/keys is outside of the scope of the proposal.

When a node sends a routing request (RREQ) it adds a top hash, a signature and a hash value. The signature certifies that all non-mutable fields (all except the hop-count) in the RREQ message were defined by this node (including the sequence number). The hash value correlates to the hop count value. For the source node, this hash value is the seed or H_0 of a hash chain. The top hash is the value H_N such that $H_N = F^N(H_0)$ where N is the maximum hop count and F is the global one-way function that is used for the hash chains. A new value H_0 is generated for every RREQ.
A node receiving the RREQ has to verify the signature and the hash hop-count so that $H_N = F^{N-i}(H_i)$ where H_i is the current hash value and i is the current hop count. A failure of any test implies packet drop. If the tests hold true the normal AODV routing update algorithm follows and the sequence number is greater than the one currently defined for this route it updates its routing table. Then the node increments the hop count of the message and the corresponding hash value. All other fields (including the signature) are left unchanged. The message is then rebroadcasted.

When the destination node receives the RREQ message it does the same verifications as the intermediate nodes. It then generates a routing reply (RREP) where it sends the same fields in the SAODV extension as for the RREQ. It sends a new hash for the hop count and a new top hash. It signs the packet with its signature. Intermediate nodes also act as in the RREQ, verifying signature and hash values as before, and only updating routing tables if both checks are valid. Hash values for the hop count are incremented as before. RREP packets are only unicast as a reverse route is now known.
Routing error messages are signed by the originating node, and only accepted by other nodes if the signature holds true. Acknowledgments of route replies are signed in the same manner.

The proposal also addresses RREP made by intermediate nodes, instead of the destination nodes. If a node already has a route to the destination node in his routing table or the RREQ allows gratuitous replies, the node can send a RREPs relaying the destination node.
To enable this, new extensions to RREQ and RREP must be defined. When a node sends a RREQ (the node which is to be the destination node of the gratuitous reply) it sends an additional signature for the expected packet of the gratuitous RREP. A lifetime for the signature is also added and signed to avoid dangling signatures and enable routes to age if not refreshed.

Later, when an intermediate node receives a RREQ for the previous node (the one that sent a RREQ with the additional signature) it can imediately generate a RREP, with the previous signature. To send the RREP it uses the signature supplied earlier. It then adds the initial lifetime. It signs the packet (where it includes the new lifetime) using its own signature.
These RREP extensions (called double signature) are treated by other nodes the same way as normal RREP. The Double signature RREQ involves an additional step of storing the signature and lifetime for later usage.

The current trend (from the previous proposals) of malicious node detection has been dropped. SAODV only cares that routes are judged with truthfull metrics, and that senders and receivers can ascertain that routing messages were really sent by the recipient stated in the packet field. Encryption of messages is not addressed (as was in the previous proposal). Signatures are used, but only from sender and receiver (the previous proposal assured signing by every node that dealt with the packet). In contrast with the previous three proposals SAODV only deals with routing packets, it does not address the security of data packets. The AODV routing metric is protected, as it cannot be decremented (it can be maintained meaning that colluding nodes can advertise a lower metric). The previous proposal secured the path advertised as it worked with a source route based protocol. This protection of the metric is more difficult to ellude as it carries the signatures of each node. The first two did not secure routing messages.
Processing capacity is alleviated through the use of less demanding operations. Signature making and verification is done only in sender and receiver respectively, freeing intermediary nodes from this process.

Advantages	Disadvantages
• Replies are dealt with sequence numbers • Metric counts are not decreased (although they can be maintained) • Routing Packets are authenticated (source and destination only, intermediate nodes do not sign packets)	• Key distribution is not considered • No encryption, it is mentioned that is meant for public ad hoc networks that do not have very high security needs • Colluding nodes can pass low metric hashes to each other • AODV lifetime (and thus validity of gratuitous RREP) is relative, not absolute, that is, a t lifetime means that a route is valid for more t msecs after the RREP was received • DoS are not addressed

5.5 # A Security Aware Routing Protocol for Wireless Ad Hoc Networks S. Yi, P. Naldurg, and R. Kravets 2002

This protocol aims at providing a path (from source to destination) with only nodes from a specified security level. It dwells only in securing the routing protocol, not mentioning data packets.

Easy eavesdropping	Use of symmetric encryption
Denial of Service	Not addressed
Identity problems	Definition of security levels Use of digital signatures Reference to usage of tamper-proof hardware
Trust issues	Use of hash signatures
Replays	Not addressed, but there is a mention to the use of sequence numbers

SAR (Security Aware Routing) follows a security clearance level hierarchy. Nodes belong to a specific level by holding a symmetric encryption key for that level, which means a key for every level in the hierarchy. This enables the encryption and hash signing to nodes of the same level.

The purpose is to obtain a path with nodes that belong to a level specified by the sender. Nodes with lower security clearance are kept outside the path. This is achieved by using the shared key of the desired level to encrypt the routing packet headers/data. In this way only nodes of that level can understand the routing information being exchanged. Messages not understood are dropped and not forwarded. The authors mention that malicious interruption of routing traffic is treated as the node being of a lower security than the one required. However if a node only drops data traffic (forwarding normally routing traffic) it could cause a black-hole, as the protocol only copes with nodes that thwart routing packets. The black hole node needs to have a valid key for the security level in order to achieve this.
Although not mentioned, we assume that nodes have the keys from the security levels lower than their clearance.

The protocol is built on top of AODV (although it could use any on-demand protocol). It adds a security level header to the RREQ and RREP messages so to specify the security level wanted, and the maximum available. This implies that the sender can request a level and find a path with a higher security. To enable this another field is added to the RREQ to maintain the highest security level possible in the path being probed.
The encryption/signing using the security level key ensures that nodes do not claim that they have greater clearance than their actual level.

The problem of node capture/take over is left to tamper-proof hardware.
Key distribution mechanisms are assumed to be in place, so the security level associated with each node is assigned by the key distribution infrastructure.
In an implementation/performance evaluation done in NS-2 [NS-2], the authors assessed that the number of routing messages exchanged is lower than in AODV, due to the dropping of higher level messages. However, message processing in nodes is expected to be high, because of the cryptographic functions used (this was not tested as NS-2 is not meant for processing measurements).

The protocol only addresses the routing packets, leaving data packets unmentioned. One can nonetheless assume that data packets (header and data) are encrypted using the same algorithm.

This protocol differs from the previous ones as it tries to establish different security groups in the network. Routing messages are secured by encryption and signing; SAODV only addressed the sender/receiver truthfullness and metric control; Resilience to Byzantine secured the path field by signing it. Here the packet vital information is encrypted with the key for the security level, leaving it inaccessible to nodes not holding the key. The advantage is that nodes not in the level can not even access what request is being made; the downsize is higher processing in each node.

Advantages	Disadvantages
• Encryption of data and headers (only routing packets mentioned) • Signing/hash digests of packets • Lower number of messages, due to nodes dropping the ones that do not belgong to their security level	• High processing in nodes as each node has to decrypt/encrypt routing messages • Trust issues are left to tamper-proof hardware • Dependence on key distribution algorithm is not addressed • No mention of data transit

5.6 # Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks
Yih-Chun Hu, Adrian Perrig and David B. Johnson 2002 ↑

This proposal aims at ensuring correct paths with route discoveries, using low computational overheads. It uses a broadcast authentication protocol developed by the authors named TESLA (Timed Efficient Stream Loss-tolerant Authentication).

Easy eavesdropping	Not addressed
Denial of Service	Route Request ratio limits and use of TESLA hash chains prevents floods of Route Requests Routing Errors must be authenticated
Identity problems	Nodes are identified
Trust issues	MACs are used in data/routing packets Keys are disclosed only when they expire, so impersonation is not possible
Replays	The temporal elements of keys mitigate replay attacks

As TESLA is an important part of the protocol we will give a brief overview of it. Ariadne can nonetheless use other methods of authentication, as digital signatures and MACs generated using shared keys.

TESLA aims at providing shared key authentication MACs without the problems associated to key distribution. To enable this, each node creates a hash chain. The hash values (K_i, $0 < i < N$) are the keys used to generate the MACs for each packet (in fact, what is really used is a K'_i derived with another one way function from K_i, this is to avoid using the same key multiple times in different cryptographic operations).

Each key is valid only for a limited time (nodes know this schedule by means of the bootstrapping phase). When a key expires the node broadcasts it to the network.
A node starts by using the K_{N-1} key. Other nodes receive the packets with the MACs and an indication of what key is being used in the packet (the index of the hash value).

Nodes know the lifetime of each key. This way, they can know if the key, used in the received packet to calculate the MAC, has already expired. If so, they drop the packet. If it is a valid key, they store the packet (including the index of the key) until the key is disclosed.
When a key is released, the nodes check the stored packets for correctness. Keys are disclosed in data packets.

The boostrapping phase is very important for determining the starting K_N of each node and the key schedule. For this, each node broadcasts K_N, the start time for the key chain (T_0), the interval duration (T_{Int}) and the disclosure delay (all this is authenticated with a digital signature or other means). A key is only valid in its interval, that means that K_i is only valid in Int_i, that starts at $T_0 + T_{Int} * i$ (this is approximate as the protocol includes network delay to achieve a loose synchronization). In each interval a node can send multiple packets (using the same key) or none at all. The disclosure delay is measured in T_{Int} units.
The initial K_N enables the authentication of received keys as $K_N = F^{N-j}(K_j)$, i.e., valid keys belong to the hash chain of K_N.
Packet losses may imply key losses, but that does not inhibit the checking of previous received packets as long as a posterior key is revealed. It is possible to calculate K_j from K_i as long as $i < j$, $K_j = F^{j-i}(K_i)$.

As can be easily seen this scheme implies time synchronization between nodes. This is loosely coupled and a Δ value is used to delay the discloser of keys.
When nodes calculate MACs they must antecipate the round trip time, so that when the packet reaches the destination it gets their before the lifetime of the key used.

The protocol Ariadne is based on DSR. As so, it has Route Request and Route Response messages.
The Request message carries the ID of the destination, source node's ID and an ID for the request. The sender calculates a MAC with a shared key between the sender and the destination (this is not a TESLA key, but a normal symmetric key that every node must share with communicating nodes). The request is then broadcasted with the expected time of arrival to the target. Each node that receives the request checks if this is a new request and if the time of arrival is still valid (the node has a key usable in the remaining time). If any of these requirements fails the packet is dropped. Otherwise, the node adds itself to a nodes list, calculates a hash of its ID and the previous hash value using a one way function ($h_i = F[ID_i, h_{i-1}]$), and a MAC of the packet with the new list. The hash value prevents the removal of nodes from the nodes list, as in the end $h_K = F[ID_K, F[ID_{K-1}, F[ID_{K-2}, ... h_0]...]]]$, where h_0 is the MAC from the source node using the shared key. However, this does not prevent the removal of every node except the source. A malicious node could remove every node and calculate a new $h_K = F[ID_K, h_0]$, because h_0 is konown.
This packet is then broadcasted. When the target node receives the request it checks the hashes from the nodes list as described, and if the keys of the MACs are still valid. It also checks the MAC from the source, using the shared key. The target then issues a response sending the full path and the MACs from the nodes list. A MAC of the entire packet using the shared key is also added. The response is now unicast to the reverse path. Each node discloses in the packet the key used in the request (waiting if necessary, for the expiration time of the key). When the sender receives the response it checks the keys returned (if are they part of the $K_j = F^{j-i}(K_i)$), that the MAC from the target is valid and that each MAC in the nodes list is also valid.

Route errors are also issued with a MAC, a disclosure key and a time for disclosure. The address of the node that encountered the error and the address of the node to which the packet was to be sent is also added to the message. The packet is sent to the previous node (following the reverse path). A node receiving this packet checks the MAC, which implies cashing the packet until the key is disclosed. Until disclosure the error is not accepted, which means that route tables are only updated after disclosure.

The proposal also mentions the use of feedback to measure path reliability. An end-to-end layer message, or a transport layer scheme is suggested to achieve this. Probes could be sent to evaluate multiple known paths to a destination, the returned ACKS would be used to measure the paths reliability.

With the objective of thwarting route request floods, a TESLA key is used in each Route Request. When the key is released the previous Route Request becomes non-valid and thus is dropped by nodes. If a schedule for the keys is also known by every node, the nodes could check if the keys should have already been released, inspite of not seing their packet disclosure. This introduces a ratio limit for route requests broadcasts.
A list of nodes to avoid in Route Requests is also considered.

This proposal tries to lower the computational efforts of node, using less demanding operations. Metric and path of routing messages are protected as in Resilient to Byzantine and in Security Aware, although it shares more resemblance to Resilient then to Byzantine. The computational effort is lighter as mainly hash functions are used, as opposed to cryptographic funtions in the previous proposals. There is also the mention of assessing path reliability as in CONFIDANT and Mitigating. The methodology is through acks from the destination instead of promiscuous hearing.

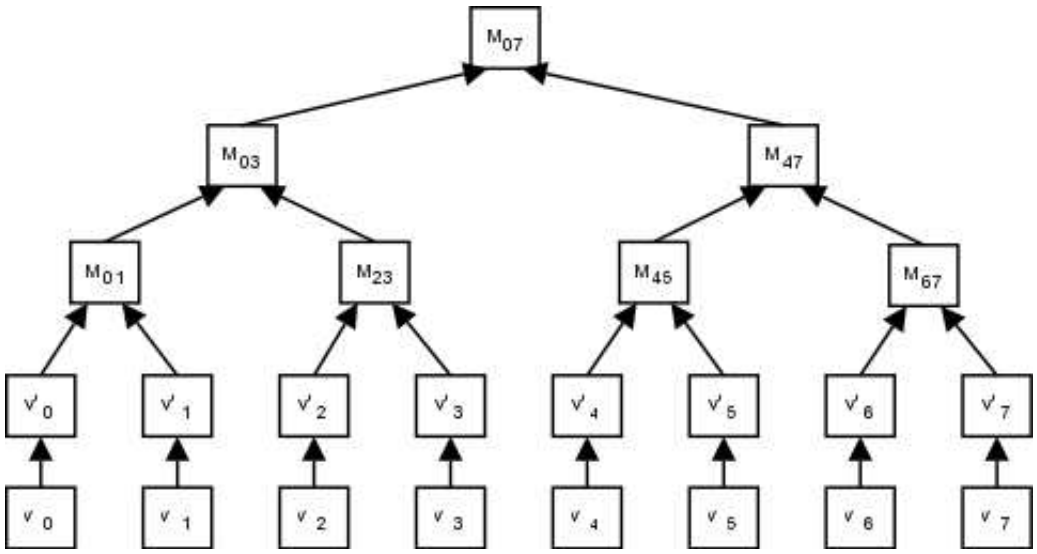
Advantages	Disadvantages
• Computational savings by using MACs with symmetric keys • Ensure truthfulness of routing discoveries and routing errors • DoS defense in route requests	• Non-repudiation is not available, due to key disclosure • Shared keys between Sender and Target must be in place • No encryption • Delays in packet deliveries (to the application layer) due to key disclosure • Route Requests ratio limits imply delays in route discovery

5.7 # Efficient Security Mechanisms for Routing Protocols Yih-Chun Hu, Adrian Perrig and David B. Johnson 2002 ↑

Through the use of one-way functions the authors aim to provide identification and proof of metric correctness in distance vector routing protocols.

Easy eavesdropping	Not addressed
Denial of Service	Techniques to avoid DoS attacks on hash calculations
Identity problems	Nodes are identified in routing updates
Trust issues	Routing metrics and sequence numbers are protected
Replays	Through the use of sequence numbers, replays are mitigated as long as nodes hear the released sequence numbers

This protocol follows the same principal as the previous Ariadne for which the authors are also responsible. Futhermore, it relies on data signing for the bootstrapping phase.
As one expects, it uses hash chains. In addition, it adds Merkle hash trees [Merkle], in which a binary tree (a binary tree is used for simplicity, other trees can also be used) is built from the leafs nodes to the root using a one-way function. Fig. 1 illustrates the process:



- $v'_i = F(v_i)$
- $m_{kj} = F((v'_k \| v'_j))$ for k-j = 1
- $m_{kj} = F((m_{k((k-j+1)/2 -1)} \| m_{((k-j+1)/2)j}$ for k-j >1
- || denotes concatenation

Fig 2 - Merkle Hash Tree

Using this tree enables the authentication of released values. First the node releases m_{07} to every node, authenticated with a digital signature or other means. When it wants to release the information 1 it sends 1, v'_1, v'_0 , m_{23} and m_{47}. These values provide m_{07} by evaluating: $F(m_{47} \| F(m_{23} \| F(v'_0 \| v'_1)))$. The only way a node could release these numbers was if it generated the tree. So the nodes that receive (1, v'_1, v'_0 , m_{23} , m_{47}) should already have m_{07} (authenticated) and check that the calculation is correct.

The protocol has a sequence number and a hop metric, to be conformant to distance vector protocols. It ensures that a node does not decrease the metric or increases the sequence number.
To achieve this, we generate a hash chain as in TESLA, discussed in Ariadne. But the values $H_i = F(H_{i-1})$ are not used as keys. Instead the node discloses H_N signed as before. It has the following constants globally defined: K, the maximum hop count possible (the infinite value in distance vector) and S, the maximum sequence number allowed in a chain. N should be S*K.
The values H_i represent the sequence number of the message and the metric. That is, the N space of H is divided in S groups for each sequence number available. In each group there are K values representing the metric values 0 to K-1.
The sequence numbers increase from H_N to H_0; in each group if i > j then H_i represents a metric higher than H_j.

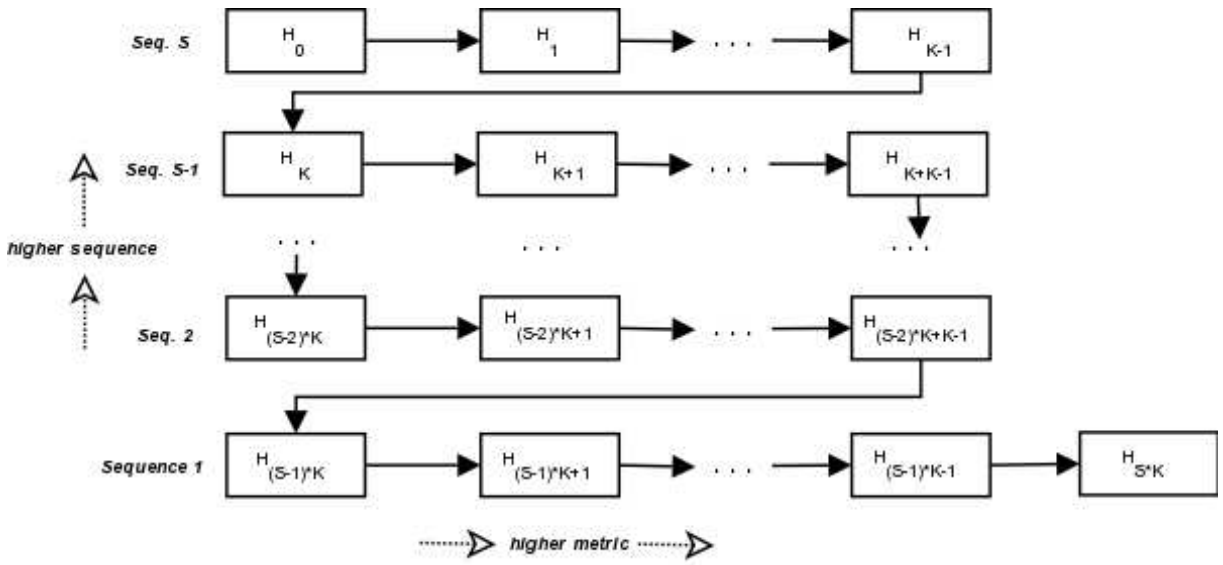


Fig 3 - Hash Chains for Sequence Number and Metric

In a message with sequence number 1 we use the H_{N-K} to H_{N-1} values to indicate 0 to K-1 metric respectively.
When sending a message with sequence number 1 a node A then broadcasts H_{N-K} indicating that it can reach itself with a 0 hop count. When one hop neighbours of A receive this, they broadcast a message indicating that they can reach A with 1 hop count, adding $H_{N-K+1} = F(H_{N-K})$. Other nodes follow the same principle always checking that the H_i received is $H_N = F^{N-i}(H_i)$ for the destination node.
This will oblige nodes to, at least, advertise the same metric (they can maintain the H_i received not hashing it).
When a node receives an update it revises its routing table if the sequence number for that destination is greater than the one he has (a fresher route) or if the sequence number is the same and the metric is lower.
In each update nodes broadcast their routing information (as for distance vector protocols) with the associated hash values (remember that these values are intrinsic to a destination node). Nodes cannot issue a greater sequence number (because it is unfeasible to reverse F()) and they cannot decrease the metric for the same reason. If nodes hear the lower metrics they probably can reach the nodes broadcasting them, thus being one hop from them. If low metrics hashes are passed by colluding nodes they can advertise better routes than the ones they really have.

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 (its discussion is beyond the scope of this writing). This construct uses Merkle hash trees to authenticate values . The construction will serve to identify the node sending the update. With this, 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 as some problems with networks with large number of nodes, as it is possible (with low probability) to overhear hash values that will enable a same distance fraud.

Regarding the protection of path information in routing packets, this paper proposes the approach used in Ariadne, but it also mentions that the use of shared keys between each node in the path could also be employed.

To increase speed performance in verifying hash values the authors introduce Merkle-Winternitz chains and skiplists. These construction will help prevent DoS, by decreasing the effort needed to verify hash chains. This aims to prevent (or at least alleviate) the effect of malicious nodes that send phony packets to force nodes to compute the hash values, testing them for correctness.
New hash chains can be constructed and their hash roots sent authenticated using Merkle-Winternitz chains.
The authors also mention other work [Jak01][Jak02] that deal with the minimization of space storage for hash chains.

This proposal is directed to distance vector protocols. As such (and being made by the same authors) it shares some common ideas with the previous one. The sequence number and hop count protection is done using hash chains in the same way as in Ariadne. Overall the protocol is very similar to Ariadne adding Hash Tree Chains to force nodes to increase hop metrics (and not just prevent its decrease), which use Merkle hash trees to authenticate values. Some efforts are also done to allow secure/authenticated distribution of hash roots and decrease computational efforts regarding hash chains which could also be applied to Ariadne.

Advantages	Disadvantages
• Use of hash functions which have lower complexity than asymmetric keys • Development of constructs to lower the complexity of authentication • Routing Updates are authenticated and with correct metrics • Computational DoS are addressed	• Hash roots need to be deployed authenticated (although after initial bootstrapping it is possible to piggyback new hash chain roots) • Metric must be hop count • Colluding nodes can pass low metric hashes to each other • Non-repudiation is not available • No encryption

5.8 #Self-securing Ad Hoc Wireless Networks
Jiejun Kong, Petros Zerfos, Haiyun Luo, Songwu Lu, Lixia Zhang 2001

This protocol strays a little from the previous ones, as it does not specifically address routing issues. Nonetheless, it focus on key issues, that the other proposals did not tackle (altough they used them).
The main objectives are to enable encryption, authentication and non-repudiation ubiquity, ensuring high availability of the key system. These cryptographic functions are all done distributedly.
The following discussion will center on the concepts, leaving the mathematical proofs aside. The interested reader is refered to the paper for further details.

Easy eavesdropping	Data is encrypted
Denial of Service	Nodes not authorized are not able to use the network
Identity problems	Packets are signed Unauthorized nodes are not accepted in the network
Trust issues	IDS are referred, but not addressed No hash mechanisms are defined
Replays	Not addressed

This proposal uses threshold secret sharing [Thresh]. It has the following pre-requisites for each node: an unique ID, that is non forgeable (or forgery is detected by IDS systems), a mechanism for local detection of misbehaving nodes (usually an IDS), at least K one-hop neighbouring nodes and a key pair for each node (public, secret key). As will be seen, the encryption mechanism uses RSA assymetric keys.
There is a global Secret Key (SK) and the corresponding Public Key (PK). SK is *'divided'* into K nodes. Each node i has a partial secret key that is a function of its ID v_i (P_{v_i}). This enables that K nodes function as a Certificate Authority using SK.
The distribution of P_{v_i} 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, but a coalition of K-1 nodes holding a partial secret share do not have any information about SK.

A node wanting to use the distributed CA must contact K nodes that have a partial secret share. These K nodes can only be one-hop neighbouring nodes, according to the specification. This is due to the fact that it is easier to collect reliable information about misbehaviour of closer nodes than multi-hop ones, which implies that information about closer neighbours is more easily gathered.
As it is expected PK is known by all nodes.

Each node must have a certificate signed by SK validating its key pair. This certificate as 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 misbehaviour, according to each node internal information. Here IDS system can be used as a way to gather knowledge about node behaviour.
Certificate renewall follows the same principles, adding that the current certificate must not be in a Certificate Revocation List (CRL). A node enters this list (which is local to each node) when the list owner, by direct monitoring, observes malicious behaviour or when K different signed accusations are received by a node. When a node observes misbehaviour it broadcasts its finding, signing the information.
Nodes in CRLs are ignored in the network. A node needs only to maintain entries in the CRL as long as the node's certificate is valid, after it has expired the node can remove the entry from the CRL, because the certificate is no longer valid, regardless of the entry in the CRL.
Nodes in CRLs have to get new certificates through offline methods.
If no information is known about a node, it is treated as a valid node.
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 responding to the node's request.

The partial secret keys are also renewed periodically. Note that SK remains the same, what changes are the partial secrets, i.e. the function that generates P_{v_i}. 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, what is necessary is that all K nodes have the same version of the function.
This scheme prevents that malicious nodes accumulate partial secret shares through updates (as long as they are identified as malicious, if they are not their partial share could also be updated).

The K nodes needed to use the SK key, can be any K nodes. This means that a node can roam to find nodes, collecting results from responding nodes. In this way mobility improves the availability of the distributed CA.

Initialization is done by an offline authority that distributes the partial secrets by K initial nodes. Nodes without a valid certificate are given one by the coalition nodes if none misbehaviour information exists (as mentioned). The authors also mention the possibility of the initial issuing being made offline or through a coalition of K nodes using colaborative admission control. The trustiness approach (give certificates and secret partial keys to unknown nodes) can lead to giving malicious nodes initial startup. The offline initialization could prevent this but would make it more difficult for new nodes to enter and move in the network.

Regarding the possibility of malicious nodes sending false partial certificates and/or generating false partial secrets for good nodes, the authors point to VSS (Verifiable Secret Sharing) [VSS1,VSS2,VSS3]. This would imply the signature of partial certificates and partial secrets with the node's partial secret. The VSS techniques (multi-signature algorithms) would enable detection of invalid partial secrets. This detours, malicious nodes with invalid partial secrets, but if valid certificates are acquired, one could surpass this. That would mean that certificate generation would fail (nodes would however discover the error before using the certificate, due to the algorithm of the generation). The partial secret generation would also fail, but only be detected (using VSS) when the node used it in a request by other nodes.

The K factor 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 done by the authors indicates that low end devices (PDAs) will be significantly delayed by the process (either as a node requester or as a part of the CA). Laptops (in the proposal PentiumIII

500) would cope with the computational work.

This proposal addresses the key distribution issues, that all others left unattended. The only common concern that can be pointed is the treatment of misbehaviour (using CRLs), which is also done in CONFIDANT. Self-Securing does not deal with misbehaviour detections (it is left to an IDS) as opposed to the promiscuous detection in CONFIDANT.
Processing concerns are stated, but the protocol depends heavily on assymetric keys, which has a heavy cost on processing power.

Advantages	Disadvantages
• Highly available distributed CA (depending on network density and security level) • Authentication, Encryption and signing through certified keys • Periodically renewed partial secrets • Certificate revocation lists • Malicious nodes are ignored (if their malevolence is detected)	• High processing in nodes (node requesters or as a part of the CA) • High dependency on IDS • Nodes in CRL must ensure out-of-bound reissuing • Initial deployment can pose a restriction in ad-hoc networks formations • Initial trustiness can pose a security problem

6 * Conclusion *

Security (or the lack of) is nowadays a growing concern in everyone's mind. Routing protocols are one of the points to be addressed. In the wired, widely used Internet routing protocols have no inherent security. These issues are now being addressed [IETF01] and some proposals exist [RoutSec] to retrofit security into these protocols.
Ad-hoc networks are only now taking their first steps, and the chance to design its routing protocols secure, instead of adding security at a later point, with all the problems that this brings, exists.
The protocols described in this report try to achieve this goal, with SAODV being the exception. Although other proposals also use as basis already developed ad-hoc routing protocols, the major constraint of not disturbing already deployed products (that the internet routing protocols encounter) is not (yet) an issue.

The two major conclusions after writing this report are that there might not be a one *'size fits all'* solution and that no current solution is self-contained.

The first statement jumps out when reading SAR and the sentence from SAODV which states "maybe it [confidentiality] is needed for scenarios with very high security needs". The fact that the security needs may vary with the circumstances is not new. However, the scope, dimension and dynamic of ad-hoc networks makes it even more appropriated to have different security (or even different protocols) in different ad-hoc networks. Groups might want to have ad-hoc networks where others (not members of the group) may not enter, and so use special protocols to enable this. SAR tries to encompass this notion, when it defines different security levels. Different groups are formed according to their level clearance. However, we might want to create an even greater barrier between groups, where they do not even share the same protocols. As can be easily perceived this leads to a very ample discussion.
From this point on when we discuss security in ad-hoc we will be meaning security where "the scenario is a public ad hoc network that everybody can join at any moment" (from SAODV). This however does not preclude the use of encryption (as in SAODV). Here we encounter another divergence. What to encrypt: everything, only data packets, only routing packets... Some proposals did not address this or left it unclear. Encryption in data packets means that keys need to exist and be at hand. If we also need them to encrypt routing packets, it is wise to correlate the usage/distribution of these keys. This common usage, is brought by the multitude of roles of each ad-hoc node. It serves as a router, but it is also a common node trying to get by, using the network for its own

purposes.

This key problem, leads to the second conclusion. Current proposals do not address the whole picture. Although many focus on multiple aspects, the one resilient to Byzantine failures has signing, HMAC, encryption and black holes avoidance; Ariadne deals with easing computational effort; Self-Securing tackles key distribution/usage; none attacks all fronts.
In our opinion the fronts are:

Signing
: to ensure information origin, non-repudiation is not (in our opinion) a must have

Integrity of messages
: using MACs or HMACs

Encryption of information
: the possibility of encryption should exist (in routing and data packets), the user/application could then define its needs

Path information/assessment
: should exist to avoid possible bad nodes

Computational/Energy savings
: a protocol should try to minimize these spendings, because low-end devices are expected to be a great percentage of nodes in this type of network

Key distribution/usage
: this is an important factor, as this must exist to enable striking the first and third front (the second could use MACs)

IDS can also be a factor to account for. The path information/assessment can be done using an IDS based technology. However, this should not grow to be a full blown IDS system, but rather the sufficient to evaluate node behaviour (in accordance to the defined factors).
The selfishness aspect dealt in some proposals (namely at Mitigating and CONFIDANT), should not be considered a requirement, because a node could use more power to evaluate the nodes behaviour than if it was imoral/unethical and forwarded all packets from the bad-behaving nodes. This does not exclude that our path information/assessment algorithms could evaluate this conduct.

In our view one should try to use the concepts depicted in "An On-Demand Secure Routing Protocol Resilient to Byzantine Failures", "Efficient Security Mechanisms for Routing Protocols" and "Self-securing Ad Hoc Wireless Networks" to be able to resist attacks in what we believe are the battle fronts of ad-hoc security.

7 * Proposal *

Steps	Added Costly Actions
Entering the Network - DSR / AODV	
1. Generate ID (using network identification) 2. Generate pk/sk if does not have one 3. Get {pk,ID} Certificate using K-coalition (Self-sec) 4. Broadcast {pk,ID,Cert} with TTL=max hops 5. Construct the first Hash Tree Chain (for hop count) (AODV) 6. Broadcast the first Hash Tree Chain(AODV)	• pk/sk generation • Other Nodes: partial cert generation • Partial Cert. aggregation • Hash Tree Chain construction(AODV)
Receiving key	
• if certificate verifies 1. add to local tables 2. rebroadcast	• certificate verification using coalition

RREQ/RREP - AODV	
1. Verify Source Sign. (hop count not signed) 2. Verify Sender Node Sign. (hop count could be signed) 3. Sign packet 4. Verify hop count (using Hash Tree Chains)(AODV) 5. Increment hop count/hash of hop count (using Hash Tree Chains)(AODV) 6. Add to routing table if conditions for update hold {In AODV nodes only cache [dest node => next hop node] which is verified by seing the source sig and the node sig, which asserts that node has received a request originating at source node}(AODV) ● In aodv we could also use RREQ for getting the pk of the node, which would be piggybacked in the RREP	● 2 signature verifications ● 1 signature ● Verify hop count using Hash Tree Chain(AODV) ● Increase hop count in the Hash Tree Chain(AODV)
RREQ arrival AODV	
1. Same as in RREQ/RREP above, except 3 2. Send Signed RREP (hop count received not signed) 3. Add Hash Tree Chain for hop count for reversal path	● Same as in RREQ/RREP above, except increase of hop count
RREP arrival AODV	
1. Verify Source Sign. (hop count not signed) 2. Verify Sender Node Sign. (hop count could be signed) 3. Verify hop count (using Hash Tree Chains) 4. Add to routing table if conditions for update hold {In AODV nodes only cache [dest node => next hop node] which is verified by seing the source sig and the node sig, which asserts that node has received a request originating at source node}	● 2 signature verifications ● Verify hop count using Hash Tree Chain
DSR	

Legend

Source Node

SSN =	`Sign(Option Type|Identification|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|Identification|Target Addr|Source Addr)`
SS =	`Sign(Option Type|Identification|Target Addr|Source Addr|CS|Hash)`
Mesg =	`Normal RREQ|SSs|CS|Hash|SS`

Intermediate Nodes

SSs =	signature from source node
CS =	encryption from source node
Hash =	`F(SIs|Node Addr|Hash_previous)`
SI =	`Sign(Option Type|Identification|Target Addr|Source Addr|HC|CS|Node List)`
Sigs =	`(SI|Sigs)`
Mesg =	`Normal RREQ|SSs|CS|Hash|Sigs`

`Option Type` = 2 in DSR 6.2
`HC` = Hop Count (could be excluded)
`Identification`= unique value (seq number)

RREQ - DSR

1. Verify Source Sign. (hop count not signed) (using `SSs`) 2. Verify Sender Node Sign. (hop count could be signed) (using top of `Sigs`) 3. New hash (create `Hash`) 4. Sign packet (new `SI`) 5. If source Sign Nonce(create `SSN`) 6. If source Crypt to dest (create `CS`) 7. Broadcast 8. Nodes can add the routes to their routing tables, verifying the Sigs List against the nodes List Prevents: ● removable of every node 'till source =>bad node unable to produce SSN (although it has SS) ● removable of previous node => bad node unable to reverse Hash (`F`) => bad node unable to produce Hash_previous (unable to produce SSN) (the security of this depends on the security of SSN production) ● removable of intermediate nodes => bad node unable to reverse CI (`Crypt_Dest`) => bad node unable to produce previous CI (Sign of Intermediate Nodes) Notes: ● If pk (for encryption or sig verification) is not known, use `get pk` ● The use of the nonce in the source node prevents the removal of every node from the node path, leaving only the source node. If the nonce was not used every node seing the RREQ would have SS, and could generate CS, because the public key of Dest is known and SSN would be SS. ● The use of F lightens the CPU compared to Crypt ● Availability of Sigs allows the update of routing tables from source to here	● 2 signature verifications ● 1 signature ● 1 hash computation ● 1 signature if source ● One Assymetric Encryption if source
RREQ arrival DSR	
1. Verify Source Sign (hop count not signed)(`SSs`) 2. Verify Nodes Sign (hop count could be signed)(`Sigs`) 3. Verify node list using Hash and CS 4. Send RREP signed, including node list (bidirectional links assumed)	● 1 + nr hops signature verifications ● 1 assymetric decryption ● nr hops has computations
RREP DSR	
1. Same as in RREQ above, except 3 {this implies a trust in the destinatination node; the source trusts the signed node path list from the destination. If extra carefulness was needed the same encryption by each node in RREQ could be done in each RREP} 2. Add to routing table (if conditions for update hold) the path from this to dest , seen in the signed full path {in DSR route table is with complete path, so only in RREP does one have certification of complete traversal by the dest node (each node signed to it), this is to prevent node removals from the path} 3. Could verify the path added in RREQ from this to source 4. In arrival no new signature is needed	● 2 signature verifications ● 1 signature (except for arrival)

Get pk for other nodes

1. Broadcast request for key of the node identified by N_{ID}. The request should carry an $REQPK_{ID}$
2. Each node hearing REQPK
   - If REQPK for N_{ID} **in** cache ignore
   - Else If {pk,N_{ID},Cert} exists, broadcast REPPK with TTL=max hops and $REPPK_{ID}$ = $REQPK_{ID}$ and the {pk,N_{ID},Cert}
   - Else
     a) rebroadcast request
     b) add REQPK for N_{ID} to cache with TIMEOUT (in AODV 6.3 = NET_TRAVERSAL_TIME, and in DSR = exponential back-off algorithm (2^request_sent*DEF_TIME (not specified, could be NET_TRAVERSAL_TIME))
3. Each node hearing the REPPK
   - If {pk,N_{ID},Cert} **known** ignore
   - Else
     - if certificate verifies (using th global PK)
       1. add to local tables
          - If REQPK for N_{ID} **in** cache
            1. rebroadcast {could rebroadcast in any case => higher key spread and higher coalition verification usage}
            2. remove REQ N_{ID} from cache if it exists
     - If TIMEOUT expired
       1. Clear REQPK for N_{ID} from cache
       2. Increase number of failures for N_{ID}
       3. If number of failures reaches MAX_FAIL (RREQ_RETRIES in AODV, not defined for DSR) return destination unreachable to application and stop REQPK sending
   - to obviate overwhelming of network resources due to unreachable nodes, the TIMEOUT is used to rate-limit sending new REQPK. A limit is imposed in the number of REQPK (as in RREQ in AODV), which triggers a destinaton unreachable to the application
   - {pk,N_{ID},Cert} will travel through the network as requests from further nodes are seen
   - Even the requesting node should broadcast the REP, to deliver the {pk,N_{ID},Cert} further (limited to TTL hops){perhaps should test do and not do}
   - timeout on REQ are used so that if no REP is seen that entry does not prevent future requests (even re-requests due to timeout of not seen REP)
   - usage of N_{ID} intends to avoid broadcasts that are unecessary. As REP are broadcast, REQ from different nodes can be answered by the same REP, thus the N_{ID} is the comon factor.
   - In REPPK only if key unknown is rebroadcast => node could have responded if asked => was not asked or already replied
   - TTL verifications are not mention, but should be undertaken before mesg arrival at this layer

Spontaneous RREP

See article
RERR
See article
Route Maintenance/Repair
See article

Our proposal tries to include the benefits of An On-Demand Secure Routing Protocol Resilient to Byzantine Failures", "Efficient Security Mechanisms for Routing Protocols" and "Self-securing Ad Hoc Wireless Networks" in a protocol that will address:

- impersonation (using sinatures)
- eavesdropping (by providing keys to enable data encryption)

8 * Acronyms *

These are the acronyms used throughout the report:

```
AODV - Ad-hoc On demand Distance Vector
CA - Certificate Authority
CONFIDANT - Cooperation Of Nodes: Fairness In Distributed Ad-hoc NeTworks
CPU - Central Processing Unit
CRL - Certificate Revocation List
DDoS - Distributed DoS
DoS - Denial of Service
DSR - Dynamic Source Routing
IDS - Intrusion Detection System
IETF - Internet Engineering Task Force
HMAC - Hashed MAC
MAC - Message Authentication Code
NS - Network Simulator
PDA - Personal Digital Assistant
PK - Publice Key
RREQ - Route REQuest
RREP - Route REsPonse
RSA - Rivest, Shamir and Adleman (public key technology)
SAR - Security Aware Routing
SAODV - Secure AODV
SK - Secret Key
TESLA - Timed Efficient Stream Loss-tolerant Authentication
```

9 * Refs *

[Jak01] Markus Jakobsson. Fractal Hash Sequence Representation and Traversal. In Proceedings of the 2002 IEEE International Symposium on Information Theory (ISIT 02), pages 437-444, July 2002.
[Jak02] Don Coppersmith and Markus Jakobsson. Almost Optimal Hash Sequence Traversal. In Proceedings of the Sixth International Conference on Financial Cryptography (FC 2002), Lecture Notes in Computer Science. Springer, 2002.
[NS-2] The Network Simulator - ns-2 http://www.isi.edu/nsnam/ns/
[IETF01] A. Barbir, S. Murphy and Y. Yang Generic Threats to Routing Protocols, April 2003
[RoutSec1] from efficient [18],[20], [21], [39], [40]
{18} - Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure Border Gateway Protocol (S-BGP) Real World Performance and Deployment Issues. In Proceedings of the 2000 Symposium on Network

and Distributed Systems Security (NDSS 00), pages 103 116, February 2000.
{20} - Brijesh Kumar. Integration of Security in Network Routing Protocols. SIGSAC Review, 11(2):18 25, 1993.
{21} - Brijesh Kumar and Jon Crowcroft. Integrating Security in Inter Domain Routing Protocols. Computer Communication Review, 23(5):36 51, October 1993
{39} - Bradley R. Smith and J.J. Garcia-Luna-Aceves. Securing the Border Gateway Routing Protocol. In Proceedings of Global Internet 96, pages 81 85, November 1996.
{40} - Bradley R. Smith, Shree Murthy, and J.J. Garcia-Luna- Aceves. Securing Distance Vector Routing Protocols. In Proceedings of the 1997 Symposium on Network and Distributed Systems Security (NDSS 97), pages 85 92, February 1997.
[VSS1] Berry Schoenmakers A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting 1999
[VSS2] Stadler M. Publicly Verifiable Secret Sharing 1996
[VSS3] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung Proactive Secret Sharing (or How to Stop Perpetual Leakage) 1998
[AODV] C. Perkins, E. Belding-Royer, S. Das RFC3561 - Ad hoc On-Demand Distance Vector (AODV) Routing - July 2003
[DSR] David B. Johnson, David A. Maltz, Yih-Chun Hu The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR) - 15 April 2003
[DSR-AODV] Samir R. Das, Charles E. Perkins, Elizabeth M. Royer Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks - 2000
[Thresh] A. Shamir, How to share a secret, Communications of ACM, 1979
[Merkle] R. C. Merkle. Protocols for public key cryptosystems. In Proceedings of the IEEE Symposium on Security and Privacy, pages 122-134, 1980.