Ariadne: Secure On-Demand Routing in Ad-Hoc Networks — an explanation for dummies

Bernd May
(bm@dv-team.de)

Seminar "Internet Routing" ,
Betreuer Cigdem Sengul, Oliver Hohlfeld, Anja Feldmann
Fachgebiet Intelligente Netzte
Technische Universität Berlin

July 14, 2009

Abstract

Wireless networks are everywhere nowadays, whether deployed as sensor networks for seismic activities in earthquake endangered areas, weather stations, or bio-aquatic chlorine measure drones in swimming pools. Security and intelligence companies use them for transmission of data from micro-sized cameras, voice phones, or wiretaps. Even students nowadays happily congregate with their laptops on conventions and during courses at their universities. Ad hoc techniques play a major part in these networks and thus grow in popularity. With the amount of actually used wireless ad-hoc networks, the question about new and secure routing techniques arises. Old methods and algorithms fail to acquire acceptable performance since they were not designed for wireless use. While many of the new proposed routing protocols work nicely in non-hostile environments, few have yet been developed to combine functionality with security. Among the proposals for this kind of combination, there is one which is named Ariadne. It uses highly efficient symmetric authentication mechanisms to reduce computing time, while keeping a low per packet overhead of additional routing data and acquiring an overall good performance. This paper will analyse the original proposal to point out its strengths and weaknesses.

1 Introduction

This paper is about discussing the secure on-demand ad-hoc routing protocol Ariadne [Hu et al., 2005]. It will give a brief introduction into wireless network routing and the basics of DSR (Dynamic Source Routing) [Johnson et al., 2007, 2001, Johnson, 1994] in section two. DSR is used as the routing groundwork for Ariadne. This background information is enhanced by section three which explains specific attack scenarios and security problems of DSR addressed by Ariadne. It also provides an overview of the assumptions made as requirement for networks to run Ariadne. Section four describes the actual security algorithms employed by Ariadne with a focus on TESLA [Perrig et al., 2002, June 2005] and the authentication of routing messages. The paper then concludes with an evaluation of the performance of Ariadne compared to optimised DSR and standard DSR without the optimisations not employed by Ariadne. This and an analysis of the security mechanisms and the pros and cons of this protocol can be found in section five. Section six finally summaries my personal thoughts about Ariadne.

2 Background

When working with wireless networks one of the first principles to keep in mind is it is a broadcast medium. This means anyone within transmission range not only the intended target, can hear messages sent. Also anyone within range can send messages. Thus as many nodes share a common medium, routing algorithms have to account for a host of new challenges. These challenges were unknown to wired networks or effected them at a far smaller scale.

High mobility of the nodes, higher frequencies of packet loss, or even unavailability of some nodes due to interference and signal mitigation in a time changing environment make older approaches like proactive routing [Lang, 2003] hard to optimise. Newer routing protocols specifically designed for wireless networks take into account the varying connectivity and highly dynamic route path changes [Feldmann and Merz, 2009a,b]. DSR is one of these protocols employing a postactive vectorbased on-demand source routing algorithm.

2.1 DSR - The Dynamic Source Routing protocol

DSR was developed in the middle of the 90s [Johnson, 1994]and has been optimised even further since then. It is a on-demand routing protocol, meaning routing information is gathered only, when it is needed. A source wanting to send a packet to a certain destination for the first time first has to discover a path to this destination to send the packet. The gained route path then is appended to every packet send into the network. It has the form of a forward sequence vector that describes the path the packet should take.

Beginning at the beginning though, this path first has to be discovered by a so called ROUTE REQUEST (RREQ) for which an Example is given in figure:1. This request is originated by the sender and contains information about the destination of the request, the originator of the request, a sequence number, and the path the request took so far (see figure:1-1). Every node that receives such a request through the broadcast medium first checks if it has already seen the packet in the past in a certain amount of time. If it has it drops the packet (figure:1-2b and 2c between B and C). The original source always drops its own route requests, which is why it is not shown in figure:1. If the packet is received for the first time, the node makes an entry in its table of recently received requests. It then appends its own identifier to the path the request took and forwards it via broadcast to all adjacent neighbours (as can be seen in figure:1-2b and 2c).

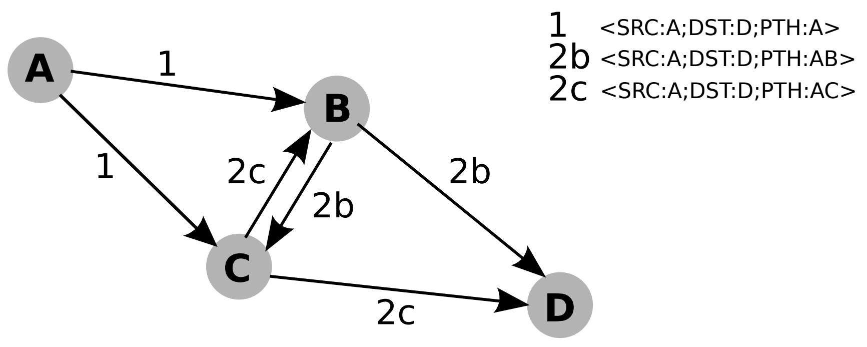

Figure 1: DSR ROUTE REQUEST

When the RREQ reaches the destination, it replies to each and every received request with a so called ROUTE REPLY (RREP). This contains the reversed path from the RREQ as a vector for the reply to take to the source of the request. Since the destination answers every request, but duplicate requests are dropped at intermediate nodes multiple paths to the same destination are possible including fully disjoint paths. The RREP is stored at the source side upon arrival, the contained sequence path vector reversed again, and used for further data transmissions.

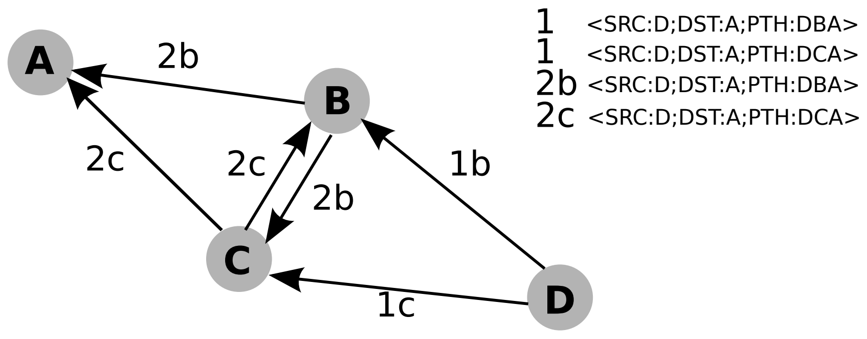

Figure 2: DSR ROUTE REPLY

Now, almost immediately the question arises, what happens if a link breaks or becomes temporarily unavailable? DSR handles this via a series mechanisms for route maintenance. Unlike for example OSPF [Moy, April 1998] though it does not rely on constant path propagation and hello messages to verify the existence of a path. In fact, DSR relies on arbitrary messages, called ROUTE ERROR (RERR) messages.

These messages are sent, whenever a node receives a packet with a sequence path vector that is no longer valid. This means the node is unable to reach the next hop in the vector, receiving no acknowledgements for the forwarded packets in a given amount of time. If this happens the RERR message is sent to the original source of the packet, containing the broken sequence path vector. This allows the originator to remove the broken route from his cache. If present in its routing table it can then choose an alternative and try again or issue a new RREQ.

Though there are further optimisations for DSR like path eavesdropping of intermediate nodes, fast route discovery by RREP packets of intermediate nodes who already know a route to the requested destination, etc, but Ariadne does not employ them. They are part of the fully optimised DSR an have distinctive impact on speed, as we will see towards the end of this paper, but describing their details goes beyond the scope of this work.

3 Attack scenarios and network assumptions

Ariadne is proposed under very specific network assumptions. Networks that differ from these assumptions might experience additional risks or lack of performance. If used with the assumptions made though, a variety of attacks and attackers can be repelled. An overview about these assumptions and possible attack scenarios on DSR will be the topics of this section.

3.1 Attackers and attacks

The authors of Ariadne propose a number of attacker scenarios and attack types, which are explained here. As a first approach attackers can be divided into two classes [Hu et al., 2005]:

1. Passive being the eavesdropping kind, this class is more a threat to privacy or anonymity of a connection and is not discussed further.
2. Active being the manipulating kind, this class eavesdrops but can also inject packets into the network of its own and change the routing behaviour.

Since Ariadne is about secure authentication of routing information only, not about privacy or encryption, further scenarios are only concerned with the active attacker type. These are then characterised by the amount of nodes inside and outside the network they control, which are further differentiated by the terms of owned (outside) and compromised (inside).

The syntax used in [Hu et al., 2005] is Active-n-m, where n is the amount of compromised and m the amount of owned nodes. The classification hierarchy (with increasing strength) then looks like this:

1. Active-0-1, the attacker owns one node
2. Active-0-x, the attacker owns x nodes
3. Active-1-x, the attacker has compromised one node and owns x nodes
4. Active-y-x, the attacker has compromised y nodes of the network and owns x nodes
5. Active-VC, a special case where the attacker has compromised one or multiple nodes on a vertex cut through the network, partitioning the good nodes into multiple sets that have to route through the compromised ones

From these attackers, different kinds of risks arise. “Attacks on ad hoc network routing protocols generally fall into one of two categories:

- Routing-disruption attacks. The attacker attempts to cause legitimate data packets to be routed in dysfunctional ways.
- Resource-consumption attacks. The attacker injects packets into the network in an attempt to consume valuable network resources such as bandwidth or to consume node resources such as memory (storage) or computation power.

From an application-layer perspective, both attacks are instances of a denial-of-service (DoS) attack.” [Hu and Perrig, 2004 May/June]
Attacks that are possible are included in this list (which by no extend is complete)

- The creation of routing loops or black holes, eventually leading to packet loss, either with or without the active attraction of data. Black holes are nodes dropping incoming packets in this case.

- A special case of the black hole, the gray hole, which attracts data and routing information but only dropping data, leading to the illusion of a working route where data mysteriously disappears.
- The partitioning of networks by controlling a vertex cut or forging routing information
- Manipulation of routing information, causing packets to use suboptimal routes and potentially pass through eavesdropping nodes. One application is the creation of a gratuitous detour, lengthening the path by adding virtual nodes.
- Blackmailing nodes in a network that uses a blacklist to identify malicious nodes. This is done by maligning good nodes, causing them to be blacklisted.
- Wormhole attacks by using virtual links between two or more nodes, effectively creating a vertex cut
- Rushing attacks by rapidly disseminating RREQ packets in networks that use duplicate suppression, causing a suppression of legitimate requests.

Ariadne proposes to solve these problems with the technique of secure authentification of messages and using MACs derived from shared secret keys to verify the integrity of the original message. To do this it heavily relies on TESLA (Time Efficient Stream-Loss Tolerant Authentication), which is explained in section 4.1. One will also see there, that additional prevention methods are necessary for some of these attacks.

3.2 Assumptions for Ariadne networks

While Ariadne does address the above issues, there are a series of assumptions made by the authors, that can affect the usability of the protocol. For example, Ariadne only works on bidirectional links. In the case of sensor networks it is not uncommon to have only unidirectional links thus limiting the application of Ariadne. They also excluded any kind of physical layer or application layer attacks, which though being beyond the scope of a routing protocol, can have severe effects on its performance. A physical jamming attack, barring any connection between any two nodes would be one of these.

Further requirements for Ariadne are the presence of a working PKI (Private Key Infrastructure) or CA (Certification Authority) and the assumptions that are inherited from the usage of TESLA (loosely time synchronised nodes, enough memory for package storage, distributed shared keys, one-way key chain computation, prediction of end-to-end transmission time, periodic time resynchronisation). The change of these assumptions or the omittance of them can have an effect on the security of Ariadne.

4 Attack prevention with Ariadne — how it works

This section will give an overview of Ariadne in action and explain its main security component, TESLA. It will also give a graphic example of the route discovery and usage of the one-way keychain.

4.1 TESLA — Time Efficient Stream-Loss Tolerant Authentication

Though MACs (Message authentication Codes) are not new to the field of authentication most algorithms heavily rely on asymmetric algorithms to implement them. Since these algorithms are computationally expensive the usage of a shared secret key seems like a viable alternative. The problem in wireless networks though, is the broadcast medium. In order for every node to be able to authenticate the MAC of a message all of them would have to share the same secret key. This would allow any node to forge packets to look like originating from the source. To address this problem of the necessity of an asymmetric primitive while using symmetric keys, TESLA [Perrig et al., June 2005] was created.

TESLA differs from traditional asymmetric functions like RSA[Rivest et al., 1978] by achieving the assymetry through clock synchronisation between nodes and delayed key disclosure. It thus has no computational expensive one-way functions to compute the hash for the MACs allowing for low work-load overhead.

The protocol basically assumes the presence of a loosely time synchronised network of nodes with a known upper bound to the synchronisation error. The sender then calculates a one-way key chain from a randomly chosen initial key by repeatedly computing a hash function on the starting value. This results in a set of keys, each of them authenticated by its predecessor, e.g. $K_i = H(K_{i+1})$.

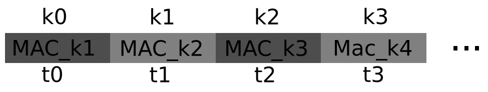

Figure 3: Keys of TESLA assigned to time slots and MACs published

The keys are then assigned in reverse order to a pre-determined schedule of time slots at which each one is published published (like in Figure:3). The first Key to be published K_0 is the one that requires a working PKI or CA to securely distribute it among all other nodes and vice versa. This is one of the starting assumptions made by TESLA. Before any node can participate in the network it has to distribute its initial key through the PKI or CA. Then it has to receive the initial keys of all other nodes from the PKI or CA. After that every key from the generated chain of a node can be authenticated by its corresponding keys that were already published, down to the initial one. This simply done by computing the hash function over a given key again and comparing it to the already published ones. Since the ability of a hash function is that its output can not be reconverted, no other node is able to produce an unpublished valid key. Concerning the schedule a simple one would be to add a constant amount of time to the start time when K_0 is published, for every interval of the schedule (so the MAC_k1 could be authenticated in t1, the MAC_k2 in t2, see ¸Figure:3).

As can be seen, TESLA relies on the fact, that a receiver of a packet can determine, whether the key for a packet has yet been disclosed or not. This is done via the upper bound of time synchronisation error and the disclosure schedule. Every node can thus compute the current time slot of the schedule and verify if a packet with a given MAC was signed with a key of the current or older slots.

If a packet is received, that is signed by a key that has already been published it is discarded. If the key is not yet published, it is stored and authenticated when the key is published. This allows TESLA to even perform in networks with large roundtrip times, as the security of the authentication is not compromised. It might lead to larger delays in packet transmission though due to large amounts of dropped packets.

4.2 Route discovery in Ariadne

The route discovery still follows the same principles as it did in DSR, with the added algorithms of TESLA in between, to authenticate every packet. Ariadne also introduces the mechanism of building a hash over the full packet at every hop. This removes the possibility of a node compromising the integrity of a packet. “Route Discovery has two stages: the initiator floods the network with a ROUTE REQUEST, and the target returns a ROUTE REPLY.” Hu et al. [2005] Ariadne provides the means to authenticate these packets and to protect the integrity of the transported data. An example of this now follows.

1. $S : h0 = MAC_{K_{SD}}(RREQ, S, D, id, t_i)$
2. $S- > * :< RREQ, S, D, id, t\_i, h0, (), () >$
3. $A : h1 = H[A, h0]$
   $M_A = MAC_{K_{A_{t_i}}}(RREQ, S, D, id, t\_i, h1, (A), ())$
4. $A- > * :< RREQ, S, D, id, t_i, h1, (A), (MA) >$
5. $B : h2 = H[B, h1]$
   $M_B = MAC_{K_{B_{t_i}}}(RREQ, S, D, id, t_i, h2, (A, B), (MA))$
6. $B- > * :< RREQ, S, D, id, t_i, h2, (A, B), (MA, MB) >$
7. $C : M_C = MAC_{K_{DS}}(RREP, D, S, t_i, (A, B), (MA, MB))$
8. $C- > B :< RREP, D, S, t_i, (A, B), (M_A, M_B), MD, () >$
9. $B- > A :< RREP, D, S, t_i, (A, B), (MA, MB), MD, (K_{B_{t_i}}) >$
10. $A- > S :< RREP, D, S, t_i, (A, B), (MA, MB), MD, (K_{B_{t_i}}, K_{A_{t_i}}) >$

Figure 4: Route discovery in Ariadne

As one can see in the first entry of Figure: 4 the source (S) creates a ROUTE REQUEST (RREQ) message containing five fields. These fields are: RREQ the message to Ariadne that this is a route request, source or initiator of this message (S), the destination towards which a route shall be found (D), the id that is used as an identifier for this specific request (id), and the time interval of the TESLA interval that is supposed to be valid when the packet arrives at the target (t_i). This five tuple is then appended by the MAC computed as a hash with the shared key (K_{SD}) between target and source (which is assumed that it was exchanged before, i.e. through the PKI). The hash is computed over the former parameters (h_0 which can be seen in step 1 and is done as soon as the first five tuple is assembled). This value is inserted at the sixth position and is the so called hash chain because other nodes will replace it with their computed hash during transfer. Two additional fields are left empty for the nodelist the packet travelled through and the MAC list of these nodes. When this eight tupelo is assembled it is broadcasted through the network (step 2 in Figure: 4 the * resembling a broadcast target)

Each node that receives this request, compares the $\langle initiator, id \rangle$ tupelo of the message with its table of last seen RREQs to prevent duplicate requests. It also checks if the key for the time interval t_i has yet been disclosed and if the time interval is too far in the future. In either positive case it drops the packet. If neither of these conditions are met, the ¡initiator,id¿ pair is stored by the node and it sets to work. First it appends its address to the node list in the request, then replaces the hash chain with the hash over $\langle A, hashchain \rangle$ and appends a MAC over the entire request to the MAC list (step 3 in Figure: 4). For computing the hash, the node uses its key from its own one-way key chain assigned to the time interval t_i specified in the request. Finally the node rebroadcasts the modified request as per DSR specification (step 4 in Figure: 4). This is then repeated at every intermediate node (steps 5 and 6 in Figure: 3).

Upon arrival at the destination, the target determines the validity of the request by checking, that the keys for the time interval have not yet been disclosed and that the hash chain field is consistent with the hash over the accumulated $\langle node, hashchain \rangle$ pairs ($H[node_n, H[node_{n-1}, H..., H[node_1, MAC_{K_{SD}} < S, D, id, t_i >$). If the request is valid a

RREP is sent to the initiator containing eight fields again. These include RREP, target D, initiator S, time interval t_i, node list (A,B), MAC list (M_A, M_B), target MAC M_C and key list. The information for the ROUTE REPLY are directly taken from the ROUTE REQUEST according to DSR, the target and initiator, time interval, node list and MAC list are also derived from the request. The target MAC is set to the MAC computed over the preceding parameters with the shared key of the $\langle initiator, target\rangle$ pair and the key list is initialised to an empty list (step 7 in Figure: 4). The REPLY is then sent to the initiator along the route indicated by the reversed sequence of hops from the nodes list of the REQUEST (steps 8, 9 and 10 in Figure: 4).

A node the receives a RREP buffers it and forwards it when it is ready to disclose the key specified by the interval t_i in the RREP. It appends its key to the key list field and then forwards it according to the source route indicated in the packet (step 9 and 10 in Figure: 4).

When the initiator receives the RREP, he verifies the validity of each key in the key list, the target MAC (M_C in Figure: 4) and of each MAC in the MAC list. If all tests succeed the RREP is accepted, if not the packet is discarded.

4.3 Route maintenance in Ariadne

While manipulation of route requests is now very difficult as we have seen, the possibility to disrupt the network with forged ERROR messages might still persist. This is also prevented by the authentication in Ariadne because it requires that RERRs are authenticated by the sender. Only known nodes are allowed to send RERRs this way.

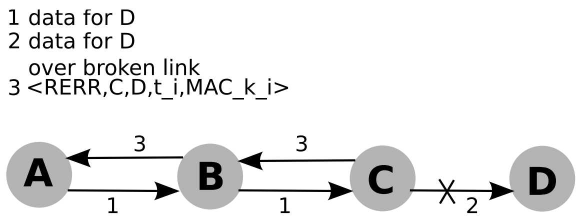

Figure 5: Route Error traversal in Ariadne

An RERR is generated as in DSR by the node that detects a broken link (node C in5). It then creates a RERR including the error (RERR), the sending address of the error node (ES), the target address the packet was destined to (node D in 5), a time interval set to the pessimistic expected arrival time of the ERROR at the source (t_i) and an error MAC field consisting of the MAC over the preceding parameters computed with the error senders TESLA key for the interval specified in the ERROR (MAC_{K_i}).

The packet is then sent along the reversed path, stored by every node that has a path using the ¡sending address, receiving address¿ pair if the time interval is valid (node B in 5) and forwarded only if the time interval is valid. The destination does the same (node A in 5). All nodes do not include the information from the ERROR until the time of key disclosure t_i. If at that time, the ERROR is found valid, the according routes are removed from cache. This especially means, that while the validation time is not yet present, the source of the original packet might generate further ERROR messages by using the old (broken) path.

5 Evaluation — does it live up to its promises?

Ariadne has many small additional mechanisms that cover a wide variety of the security issues mentioned in section 3, but it also comes with a few drawbacks and additional complications. These affect performance and computation time but also open up some other security issues that have to be dealt with.

5.1 Performance, comparing DSR to Ariadne

While Ariadne does not use some of the optimisations proposed for DSR the performance tests show, that it is in most cases more stable than DSR without them (see Figure: 6). For example average latency outperforms non-optimised DSR by about 40-50%. Packet delivery ratio is about 5% higher on average and the packet overhead is also around 50% lower than non-optimised DSR. Concerning byte overhead Ariadne is about 5 -25% higher than the according DSR. Especially the latency and packet delivery ratio is induced by the fact that due to the delay between receiving and authentication of the requests and sending the reply, short-lived routes are not traversed in Ariadne. This leads to the establishment of more stable routes in less mobile environments. The data also indicates, that in highly mobile environments with rapidly changing routes Ariadne still performs reasonable well compared to an equally inhibited DSR. Also this shows that Ariadne does not suffer from high performance loss due to security, achieving the goal of low impact on the connection.

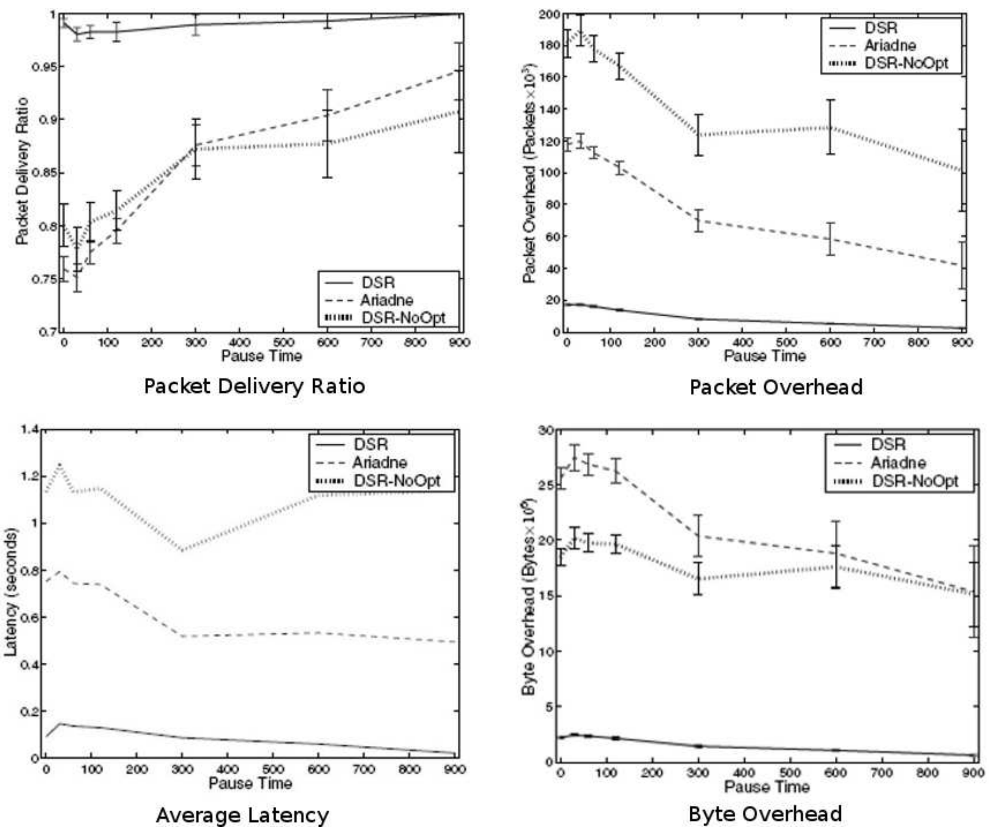

Figure 6: Statistic overview of Ariadne performance

5.2 Security analysis

Ariadne does in fact address all the issues mentioned in section 3. Active-0-1 are inhibited in all kind of attacks by Ariadne as they posses no valid key inside the network. Active-0-x attackers can at most perform wormhole and rushing attacks since they can not manipulate any packets. Packet leashes as proposed in Hu and Perrig [2004 May/June] can prevent this and are mentioned in the proposal of Ariadne but not covered by the protocol itself.

Active-1-1 attackers could try to manipulate RREQs using the compromised node, but since the RREQ is secured by a per-hop hashing this tampering is prevented. This effectively removes the threat of black and gray holes. They could also flood the network with RREQ. Since every RREQ has to be signed and only one node is compromised, all RREQs would have to be signed by the compromised node. An upper bound to RREQs can prevent this but allows to get other nodes to blacklist the compromised node. The attacker could also try to perform a rushing attack, which is not addressed by Ariadne itself.

Active-1-x attackers might additionally try to perform wormhole attacks. Though Ariadne itself does not prevent these attacks, the authors propose the use of packet leashes Hu and Perrig [2004 May/June] and GPS data to reduce the problem to one of an Active-1-1 attacker or even completely defend against this kind of attack.

Attackers that have compromised multiple nodes (Active-y-x) while owning multiple nodes may additionally lengthen the route. Though a node will likely chose a shorter route if it finds one, Ariadne does not address the problem if there is no shorter route advertised. The Active-y-x attacker might also force a RREQ initiator to repeatedly request again, by constructing a path through its compromised node network and subsequently sending RERR messages when the path is to be used. Again Ariadne does not directly address the issue but proposes a possible solution, by including data into the RREQ.

Concerning attackers that control a vertex cut (Active-VC) he may perform additional attacks. For example make nodes on one side of the cut believe that a node from the other side is flooding the network. This is done by holding the RREQs from the node for some time and then issuing many route discoveries with the chain values from the old discoveries. With the use of time synchronised elements in the route discovery chain, these elements would just be discarded. Still flooding and blackmailing are possible. The VC attacker is also able to create a grayhole by only forwarding RREQ and RREP packets. This issue is not resolved in Ariadne.

An additional threat that arises from the use of Ariadne is the possibility of a RERR overflow, since all RERRs are stored at each node that has parts of its path.

5.3 Pros and cons

Ariadne provides means for secure routing information authentification and manipulation protection. It does so, with the use of symmetric cryptography, low computational and data overhead in messages(see 5.1) and provides a time changing resistant asymmetric principle with TESLA.

Drawbacks include lowered performance compared to classic DSR, the necessity of a loose time synchronisation between nodes, a working PKI, certification authority or pairwise shared secret keys between every node. Also though the authentication is not compromised as long as the nodes are not compromised, Ariadne does not fully secure the ad hoc routing. Additional means like linked packet leashes with GPS, blacklists and data combination with RREQs are required for specific attack scenarios. The calculation of TESLAs one-way key chain takes computation time and storage space, which should be taken into account also when working with nodes who are restricted in either way.

Additional questions arise about the possibility of the RERR floods. These is especially volatile if the attacker has compromised more than one node, as each additional node allows for more, signed RERR messages. Though Ariadne proposes to use an $\langle RERR, id \rangle$ list at every node to drop duplicate RERRs, a compromised node could just rapidly switch the RERR ids. An upper bound for RERR messages as for RREQ could reduce the problem, but at the cost of nodes being blacklisted that might be good nodes and only forwarding the RERR messages.

A final drawback is inside the one-way key chain of TESLA itself. Since the keys have to be assigned to time slots before being used and the key chain is finite, an ongoing connection might run out of keys. By assigning the keys to time slots instead of packets one gains a more packet per key ratio at the cost of end-to-end delay but it only alleviates the problem. There are proposals to how this can be solved [Perrig et al., 2001] but they are not part the original TESLA [Perrig et al., 2002].

6 Conclusion

While the authors of the Ariadne paper [Hu et al., 2005] prove that Ariadne provides secure authentication in a very fine grained way, it is noticeable that there is still much work to do. The solutions proposed against individual attack scenarios require further research and implementation. Most of these solutions where not even implemented in Ariadne itself. Some of them should also be seen in a wider context. For example when regarding the packet leash solution combined with GPS, one should note that this might provide an additional risk to privacy. Though one should keep in mind that Ariadne does not strive to achieve privacy it is a desirable goal in future routing protocols. Other parts of Ariadne open up new security risks like flooding or leaving stale packets inside the networks, never to be authenticated. Because nodes buffers packets with a time interval in reasonable future, they might run out of buffer space if a malicious node would send them a lot these packets. The problem with ongoing connections due to TESLA is also something that should be taken into account. Sensor networks with continuous data streams might experience problems here because TESLA requires one to know the amount of time for which keys are generated. After that a new initial key for a new key-chain has to be reseeded from the PKI or CA.

The paper thus provides a good start for the field of routing security, but in my opinion is still lacking. Before deploying a secure routing protocol like Ariadne, the above issues should be fixed and are subject to further research.

References

Anja Feldmann and Ruben Merz. Wireless routing in the beginning. Technical report, June 2009a. URL `http://www.net.t-labs.tu-berlin.de/teaching/ss09/IR\_lecture/`. last seen on 08.06.2009.

Anja Feldmann and Ruben Merz. Introduction: Wireless routing. Technical report, June 2009b. URL `http://www.net.t-labs.tu-berlin.de/teaching/ss09/IR\_lecture/`. last seen on 08.06.2009.

Yi-chun Hu and Adrian Perrig. A survey of secure wireless ad hoc routing. *IEEE SECURITY & PRIVACY*, 04:28–39, 2004 May/June.

Yi-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. *Wireless Networks (WINET), ACM and Springer*, 11: 21–38, 2005.

D. Johnson, Y. Hu, and D. Maltz. The dynamic source routing protocol (dsr). *IETF mobile ad hoc networking Working Group INTERNET DRAFT, RFC*, 4728, 2007.

David B. Johnson, David A. Maltz, and Josh Broch. Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks. Technical report, Computer Science Department Carnegie Mellon University Pittsburgh, PA 15213-3891, 2001.

DB Johnson. Routing in ad hoc networks of mobile hosts. In *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, pages 158–163, 1994.

D Lang. A comprehensive overview about selected ad hoc networking routing protocols. Technical report, Technische Universität München, Department of Computer Science, March 2003. URL `http://www.bib.informatik.tu-muenchen.de/infberichte/2003/TUM-I0311.pdf`.

J Moy. Ospf version 2. Technical report, RFC 2328, IETF, April 1998. URL `http://www.ietf.org/rfc/rfc2328.txt`.

A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient and secure source authentication for multicast. *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2001)*, pages 35–46, February 2001 2001.

A. Perrig, R. Canetti, JD Tygar, and D. Song. The TESLA broadcast authentication protocol. *RSA CryptoBytes*, 5(2):2–13, 2002.

A. Perrig, D. Song, R. Canetti, JD Tygar, and B. Briscoe. Timed efficient stream loss-tolerant authentication (tesla). Technical report, RFC 4082, IETF, June 2005.

RL Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.