NTNU

Norwegian University of
Science and Technology

Implementing a Secure Ad Hoc Network

Espen Grannes Graarud

Master of Science in Communication Technology

Submission date:	June 2011
Supervisor:	Stig Frode Mjølsnes, ITEM
Co-supervisor:	Martin Gilje Jaatun, SINTEF
	Lawrie Brown, UNSW@ADFA

Norwegian University of Science and Technology
Department of Telematics

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING

PROBLEM DESCRIPTION

Student:	Espen Grannes Graarud
Title:	Implementing a Secure Ad Hoc Network

Description:

B.A.T.M.A.N. is a pro-active routing protocol for ad hoc networks that is designed to be a simpler and more robust alternative to the OLSR protocol. The project work of Anne Gabrielle Bowitz and Espen Graarud proposed to add routing message authentication in B.A.T.M.A.N. using proxy certificate mechanisms.

This thesis work will determine the performance of this secured protocol in comparison with the original B.A.T.M.A.N protocol, for instance by measuring network convergence time. Tests should also be run to find how the secure protocol protects against known attacks such as the wormhole attack.

Deadline:	June 29, 2011
Submission date:	June 29, 2011
Department:	Department of Telematics
Supervisor:	Stig Frode Mjølsnes, ITEM
Co-Supervisor:	Martin Gilje Jaatun, SINTEF
	Dr. Lawrie Brown, UNSW@ADFA

Trondheim, June 29, 2011

Stig Frode Mjølsnes, NTNU ITEM

Abstract

In emergency situations such as natural disasters the emergency personell should be able to establish communication fast and reliably. Depending on the nature of the disaster one cannot rely on existing communication infrastructure, or access to centralized administration. Additionally the established communication needs authentication in order to handle access control so only trusted parties can partake. A suitable medium for such communication is wireless ad hoc networks, but their flat structure make authentication a very challenging task.

In this thesis a system design for an ad hoc routing protocol combined with access control is proposed, and implemented extending a popular routing protocol called BATMAN. The proposed authentication scheme relies on special public key certificates called proxy certificates, and combined with a neighbor trust mechanism both authentication and access control are managed in a secure manner.

Tests using mobile nodes running this implementation shows that the performance of the proposed design is comparable to the original routing protocol (BATMAN) used, and that the authentication process is manageable even for mobile ad hoc networks.

Preface

This master thesis is written by Espen Grannes Graarud and concludes my 5 year master programme in Communication Technology specializing in Information Security at the Norwegian University of Science and Technology, NTNU.

This thesis is a continuation of my and Anne Gabrielle Bowitz' Information Security specialization project that was proposed by Dr. Lawrie Brown of UNSW@ADFA, Australia, and Martin Gilje Jaatun of SINTEF ICT, Norway.

I would like to thank my fellow student Anne for our great co-operation during the project and for her thoughts and ideas during the writing of this thesis. I would also like to thank my supervisors, especially Martin for his great weekly feedbacks which was a great help along the way.

Finally I would like to thank my responsible Professor Stig Frode Mjølsnes from the Department of Telematics at NTNU for his feedback on the system design.

Trondheim, June 29, 2011

Espen Grannes Graarud

Acronyms

3G 3rd Generation Mobile Telecommunications

AL Authentication List

AM Authentication Module

BATMAN Better Approach To Mobile Ad-hoc Networking

CA Certificate Authority

CBC Cipher-block Chaining

CRC Cyclic Redundancy Check

CRL Certificate Revocation List

DHCP Dynamic Host Configuration Protocol

DoS Denial-of-Service

ECC Elliptic-Curve Cryptography

EEC End-Entity Certificate

IV Initialization Vector

LLPKC Long-Lived Public Key Certificate

MAC Message Authentication Code

MANET Mobile Ad Hoc Network

NL Neighbor List

OASIS Open Advanced System for dISaster and emergency management

OGM Originator Message

OLSR Optimized Link State Routing

OSI Open Systems Interconnection

OTP	One-Time Password
PC	Proxy Certificate
PC0	Proxy Certificate 0
PC1	Proxy Certificate 1
PKI	Public Key Infrastructure
SP	Service Proxy
SSO	Single Sign-On
TTL	Time To Live
Wifi	Wireless Fidelity (See '802.11' in Definitions)
WOT	Web Of Trust

Definitions

802.11 IEEE 802.11 standards for wireless computer networks on the 2.4, 3.6, and 5 GHz frequency bands.

Ad Hoc Network A self-organizing network with no form for pre-existing infrastructure or centralized administration.

Asymmetric Link If traffic is only possible in one direction, i.e. a node can receive but not send packets to another node, the link in between them is called an asymetric link.

Authenticated List A list containing the public keys, IP, roles, certificate validity period, of all known and authenticated nodes in the network.

Authentication The process of verifying an alleged identification.

Authentication Module Addition to the B.A.T.M.A.N. protocol which takes care of cryptographic functions and other additions. It also adds fields to the Originator messages which can contain a digital signature or signature fractions, and sends other messages with nonces, certificates, and ALs.

Authentication Token Is something that can help the authentication process. This could be e.g. a public key certificate or a smart card etc.

Authorization The process of deciding which rights, or access to which resources, an authenticated identity can have.

Certificate Authority An entity that issues certificates in a Public Key Infrastructure (PKI).

Congestion A state in wich the the amount of traffic on a network surpasses the stable amount of traffic the network can handle. I.e. congestion can make the network useless if not handled by some control mechanisms.

Convergence Time The time it takes for the network to get to a stable state with no route flapping after an event that has changed the network topology. E.g. a node has died or moved and made a link inferior to other alternative links.

Elliptic-Curve Cryptography Public key cryptography based on the mathematical properties of elliptic curves.

End-Entity Certificate A X.509 public key certificate of an end user. EECs cannot be used to sign and issue other certificates, with the exception of Proxy Certificates.

Ephemeral Key A temporary symmetric encryption key.

Keystream-material message A message containing an encrypted ephemeral key, IV, nonce, and a digital signature used to generate the sender's keystream.

Link-local See Neighbor.

Neighbor Neighbor refer to a direct neighbor, i.e. a node within transmitting range for which you can communicate directly with.

Originator Synonym for a Batman interface which is a network interface utilized by Batman.

Originator Message Batman protocol message advertising the existence of an originator. They are used for link quality and path detection [NALW10].

Packet Delivery Ratio Amount of packets received divided by the number of packets sent.

Pro-Active Routing Protocol A routing protocol that regularly broadcasts routing announcements and forwards routing announcements it receives, actively discovering routes before (pro-active) they are needed for data traffic.

Proxy Certificate A X.509 certificate signed by a regular X.509 EEC. It is used to assign roles to which the recipient can act on behalf of the signee.

Proxy Certificate 0 A proxy certificate belonging to a Service Proxy, able to issue new proxy certificates (PC1), delegating its rights to the receiver of that proxy certificate.

Proxy Certificate 1 A proxy certificate belonging to regular trusted nodes, not able to issue other proxy certificates.

Public Key Infrastructure Every entities involved with the management (creation, distribution etc.) of public key certificates. Managed by the PKIX working group of IETF.

Route Flapping Occurs when a node in a network continuously changes preferred route between a source and destination pair creating route instability.

Routing Protocol A protocol that finds and creates paths, or routes, to other nodes in the network.

Service Proxy Master node controlling the authentication and authorization functions in the network. The SP has a PC0 issued by its regular long-lived public key certificate.

Socket UNIX file descriptors or logical interfaces used to send and receive data over a network interface.

Thread A separated program flow sharing memory with its parent process. Used to utilize multiple CPUs or CPU cores, or to have more than one thread running tasks at the same time.

Web Of Trust A decentralized trust model where trust of a node is established if your trustees trusts that node.

X.509 Certificates Standard public key certificate standard managed by the PKIX working group of IETF.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We have become accustomed to an almost complete presence of digital networks in our daily lives. Everywhere you go, you can either plug your laptop into an ethernet slot, connect your iPad to an available wifi hot spot, or just use your cell phone via 3G mobile data network. However, this is not universally true throughout the world. Many places are sparsely populated, or the people living there do not have the resources to deploy such networks.

In emergency and/or military situations, this often applies. Even if it didn't, the networks may have been put out of operations due to the nature of the emergency (i.e. tsunami destroying the infrastructure). As recent events here in Norway have shown, internal errors might paralyze the whole network infrastructure[1] making emergency relief ineffective[2], creating a sound argument for having a separate backup emergency network. The military might also be in an hostile environment where they cannot trust the network in place altogether.

Emergency search and rescue and military tactical operations can greatly benefit from the use of digital communication for sharing operation critical information. If they have no trusted data network available, they should therefore set up one themselves. A realistic approach would have to be easy and quick to set up and be self-managing, thus requiring minimal maintenance. It should to an extent always be available to the participants wherever they go, which calls for using a wireless network. Last but not least, the network needs to be trusted, i.e. you should trust that the infrastructure is not compromised and that the communicating parties on the network are who they claim to be.

A Mobile Ad Hoc Network (MANET) solves some of these requirements. It does not need an existing communication infrastructure, it is self-organizing and the network coverage range can easily be extended by placing intermediate nodes in strategic

[1]http://www.dagbladet.no/2011/06/16/nyheter/innenriks/telenor/16942385/ (Norwegian)
[2]http://www.dagbladet.no/2011/06/11/nyheter/ver/flom/naturkatastrofer/innenriks/16880835/ (Norwegian)

locations. The latter requirement however, is a more challenging task in MANETs.

With the lack of infrastructure in MANETs and no guarantees that they are connected to the Internet, establishing trust between the nodes becomes different from how this is done on the Internet where we can rely on e.g. Public Key Infrastructures (PKIs).

In this thesis I will propose and implement a solution suggestion to establish a trust mechanism, i.e. an authentication scheme, which combines features of a typical PKI with some of the ideas behind Web Of Trust (WOT) [Zim95]. The system design is presented in two parts, the part which has been implemented in Chapter 3, and the ideas that I did not have time to implement are discussed as further work in Chapter 6.

As one might expect, it is a very challenging task to achieve strong security for MANETs and still have the benefits of its simple "plug and play" design. As real world implementations go, there are a few trade-offs, and security cannot always win. This design will not try to be 100% secure, but should be secure enough to deploy in emergency situations. To back this claim, the Norwegian Army recently stated that their new computer security guidelines is to rather have a usable (available) system which might be open to attack, instead of a bad system which is impenetrable - as long as they are able to monitor and take action against potential attacks[3].

1.1 Motivation

The 7.0 magnitude earthquake that struck Haiti in 2010 showed us how huge relief efforts easily become very inefficient when huge amounts of emergency relief personnel work at a scene with little or scarce communication throughout the area[4]. With a trusted communication network like a secure MANET an operation like this could become much more efficient, bringing the right amount of help to the right places and at the right time.

1.2 Contributions

This thesis presents a novel design to achieve authentication and trust between nodes in a secure ad hoc network. The popular ad hoc routing protocol called BATMAN has been extended to become an instantiation of said design, which has never been done before. Additionally, the use of proxy certificates for trust establishment for ad hoc networks is also a novel approach to the problem.

[3] http://www.tu.no/it/article287598.ece (Norwegian)
[4] http://www.wired.com/magazine/2010/04/ff_haiti/

1.3 Objectives

The main objective of this thesis is to design and implement an authentication extension to ad hoc networks based on a known routing protocol.

Secondly, other design ideas, or things that was supposed to be in the design but did not make the time frame is discussed upon in contexts of both security and real world performance.

Last, but not least, testing of the proposed design's implementation should be done to compare the performance of the new implementation against the original routing protocol.

The problem description also mentions testing the implementation against known security attacks. However, my responsible Professor Stig Frode Mjølsnes claimed no such tests were necessary as the security of this design should rather undergo peer review and testing, therefore these tests have not been done.

1.4 Limitations

1.4.1 IP Address Configuration

Autoconfiguration of network interfaces for ad hoc networks is a huge and difficult task and will not be addressed in this thesis. Throughout this thesis the assumption is that all nodes trying to participate in the same network is pre-configured with a valid and unique IP on the correct subnet. It is also assumed their network interfaces are correctly set up to connect to the correct wireless channels.

1.4.2 Detecting malicious behavior

One attack vector which will not be discussed in this thesis is if a legitimate node acts maliciously, which might happen if the private key of a legitimate node is compromised. The solution proposed in the thesis assumes all trusted nodes acts with good intentions. There have been much research about detecting malicious behavior in ad hoc network [PM04] [DOG$^+$].

However, these kind of solutions are mainly designed for networks without any authentication scheme at all, and is therefore just investigating malicious behavior without trying to detect whether a node is compromised or not. These proposals might therefore not be of the greatest interest, but should be studied to see if any of their features can safely be applied to an ad hoc network with an authentication system in place.

1.5 Method

The primary research method conducted in this thesis is the *design science paradigm* for Information Systems research as described in [HM03]. The model and method artifacts of this paradigm are described in Chapter 3 whereas the instantiation artifact is described in Chapter 4.

Much of the design (method artifact) comes from the specialization project last fall [BG10], but some aspects of that design has been changed during the course of the study and implementation in this thesis.

1.6 Document Structure

This thesis report is structured as follows:

Chapter 2: Background aims to give the reader the necessary insight about the technologies, ideas and theories discussed later in this thesis.

Chapter 3: System Design proposes an original solution for an authentication scheme for MANETs.

Chapter 4: Implementation presents the implementation of the system design. The implementation is a modification of the Better Approach To Mobile Ad-hoc Networking (BATMAN) source code.

Chapter 5: Testing & Results devise different tests for checking the performance of the implementation compared to the original BATMAN implementation, and presents the results of the tests.

Chapter 6: Discussion looks at some of the possible vulnerabilities in the proposed design, talks a little about the experience of implementing such a system, and takes up issues regarding extending the proposed system design even further.

Chapter 7: Conclusion makes conclusions about the security, and performance of this system as well as how well it fulfills the requirements for the implementation.

Appendix A: Source Code shows a few of the most necessary code snippets and links to the full source code available online.

Appendix B: Lab Setup shows how the machines used in the lab and tests were set up.

Appendix C: Test Results presents the numerical results and the logs produced in the tests.

Appendix D: Scientific Paper about adding security to the BATMAN protocol written by myself, Anne Bowitz, and our supervisors Martin Jaatun and Dr. Lawrie Brown.

Chapter 2

Background

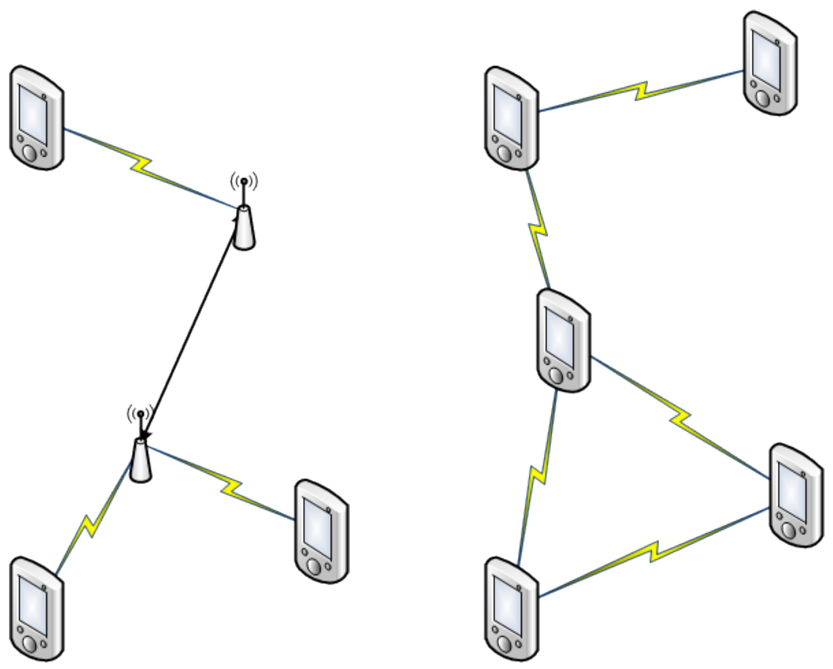
This chapter aims to give the reader the necessary background to ad hoc routing, attacks on ad hoc networks, and about the proxy certificates concept in order to understand the concepts and discussions of this thesis.

2.1 Mobile Ad Hoc Network

An ad hoc network is a network of nodes which communicate with each other and through each other, i.e. nodes act as both regular end nodes and as intermediate nodes, similar to routers in regular networks. Because nodes in an ad hoc network can route packets from one node to another until its destination, ad hoc networks do not need an infrastructure with routers such as regular networks do [Per08]. Without an infrastructure ad hoc networks can be established spontaneously and with no, or minimum, pre-configuration depending on the implementation in use.

Figure 2.1 depicts the difference between a regular wireless network and an ad hoc network. In 2.1a there are three mobile nodes connected to each other through two wireless access points (infrastructure), while in 2.1b there are five nodes connected to each other and through each other. In infrastructure mode an end node only accepts packets addressed to itself, while in ad hoc mode the node will try to forward packets addressed to other nodes (on the network layer). They will however not accept packets sent to other link-layer addresses.

A Mobile Ad Hoc Network (MANET) is a subset of ad hoc networks, in which nodes are specified as non-stationary wireless nodes, i.e. a MANET consists of several mobile nodes communicating with each other, and through each other even while on the move. With MANETs nodes are always on the move and must therefore update their routing tables continuously to make sure full paths between nodes are updated as quickly as possible, i.e. a minimal routing path convergence is necessary.



(a) Infrastructure Mode (b) Ad Hoc Mode

Figure 2.1: Difference between a regular infrastructure network and an ad hoc network

In this thesis the terms “network”, “MANET”, and “ad hoc network” are used interchangeably meaning a “MANET” as described in this section unless otherwise specified.

2.1.1 Routing

There are many routing protocols for ad hoc networks and some of the popular protocols are OLSR [CJA$^+$03], B.A.T.M.A.N. [NALW10], Babel [Chr11], DSDV [He02], and AODV [PBRD03] which all fall into one of the two main categories of ad hoc routing - i.e. “reactive” and “pro-active” routing protocols. In this thesis a pro active routing protocol is used in the system design, and as such an introduction to this type of protocol is necessary.

Pro-active Routing Protocols

As the name suggests, a pro active routing protocol finds or creates routing paths before, i.e. pro actively, the paths are requested. Essentially this means that the routing protocol on a node has to regularly share routing information with other nodes it can communicate with. This can be done in a huge variety of ways - from storing full paths to all known nodes and broadcast this information to all nodes regularly, to only storing a first hop in the direction of the known nodes, and only telling your neighbors that you have a route to some other node instead of the full route.

Each method might have some advantages over the other, i.e. storing full paths and sending all this information to all nodes in the network might minimize the chances of routing loops and route flappings, but this method would have a huge overhead in signaling traffic. The latter method would have much less overhead, but it is more difficult to avoid routing loops and/or route flapping.

Reactive Routing Protocols

In opposition to pro-active protocols, reactive protocols calculates and updates routes when they are needed, i.e. when data is being sent. The advantages of this strategy is conserving power used in obsolete/non-necessary routing calculations, but the downside being that the protocol might react more slowly to topology changes.

2.1.2 Challenges

There are a multitude of challenges to solve in regards to ad hoc networking, and especially so in mobile ad hoc networking. First there are challenges regarding routing such as route flapping and routing loops, then there is the issue of power conservation as mobile ad hoc nodes usually runs on battery power, and last, but definitely not least, there is the issue of security.

Route Flapping

Route flapping is a term describing rapid route changes when there are multiple possible routes between two nodes. Sometimes two routes might be almost equally good and might stay that way for a while, which might lead routes in a node's routing table to constantly change between the two. In such scenarios a routing protocol might solve the problem by not changing routes unless the other route is a significantly better route, or if it has been slightly better for a longer period of time. Which is the better solution is difficult to determine and is one of many reasons why there are a huge amount of contesting ad hoc routing protocols, and this question remains unresolved.

Routing Loops

You have a routing loop if a path between two nodes passes through one or more node(s) on the route more than once. When nodes have wrong, incomplete, or just different routing information from one another then two nodes in the network might choose to route the same packets through different paths, which might lead the route to eventually return to nodes which have already received and forwarded said packets.

In a MANET where routes might change very often, the chances of two nodes having different routing information is high. Routing loops can therefore become a severe problem in MANETs, suggesting implementations should use a protocol that has some mechanisms to minimize the chances of routing loops, and/or detecting routing loops and removing them after the fact.

Power Conservation

With mobile nodes in a MANET power consumption can become a huge hindrance. Because pro active routing protocols sends routing information regularly, even if they are not needed, they might be wasting energy compared to reactive protocols that only calculates routing paths when application data is sent through the network.

This issue will also be very much affected by the security implementation on top of the ad hoc routing protocol. If. for example, every packet is to be digitally signed, a hashing operation and an public key encryption operation would be performed on each packet - using a lot of energy. When designing the authentication solution this issue needs to be addressed.

Security

Security can be divided in multiple fields of interest such as (but not limited to) *confidentiality*, *integrity*, *authenticity*, and *access control* to name a few. Whatever the security feature is, it usually depends on *authentication*.

Achieving a secure initial authentication is a very difficult task even in regular networks, but using a verifiable identity based upon this initial authentication is conquered using Public Key Infrastructures (PKIs). In a MANET, possibly without Internet access, this task is again very difficult. Therefore, one ends up with the original problem - doing initial authentication all over again. As this is topic is greatly discussed throughout the thesis there is no point in elaborating more at this point.

2.2 B.A.T.M.A.N.

BATMAN [NALW10] is an increasingly popular routing protocol for wireless ad hoc networks, as seen by the fact its taken into the Linux kernel net tree[1]. The name is an abbreviation for "Better Approach To Mobile Ad hoc Networking". The motivation behind developing BATMAN was to replace the Optimized Link State Routing Protocol (OLSR) [Mes10] because of the inherent difficulties that protocol has, as explained below.

2.2.1 From OLSR to BATMAN

OLSR is a pro-active routing protocol, which means that participating nodes regularly exchange routing information with each other. According to the BATMAN developers the problem with OLSR is that every node in the network calculates the whole routing path, which is a complex way to do it. Not only is it difficult to make sure all nodes have the same information at the same time, it also needs (relatively) much storage and computation time. If nodes sit on different routing information this concept leads to routing loops and heavy route flapping. The result is many patches to the protocol that defies the protocol standard in order to make it more suitable [Mes10].

The BATMAN developers therefore wanted to start with a clean slate. They decided amongst other things that each node should only know the next hop, i.e. the link-local neighbor that is the path between itself and the destination. In many ways, what they did was to make a simpler and easier to understand protocol. For instance, the way BATMAN calculates the optimal route, i.e. the next jump, is by comparing the number of routing messages it has received from each node and who was the last sender.

2.2.2 BATMAN Protocol Explanation

The routing messages sent in BATMAN are called Originator Messages (OGMs). Figure 2.2 shows the packet format with all header fields. The OGM format has changed since the draft specification was published [NALW10], but there is no official publication with the new packet format as of yet. The updated packet format can be found in the project's internal documentation[2]. The packet format found in the draft specification belong to the older version III of the BATMAN algorithm. The algorithm used in this thesis is version IV.

The real workhorse of the packet is the "Originator Address" field which carries

[1]http://www.open-mesh.org/wiki/open-mesh/2011-03-17-batman-adv-and-the-penguin
[2]http://gitorious.org/batman-adv-doc/

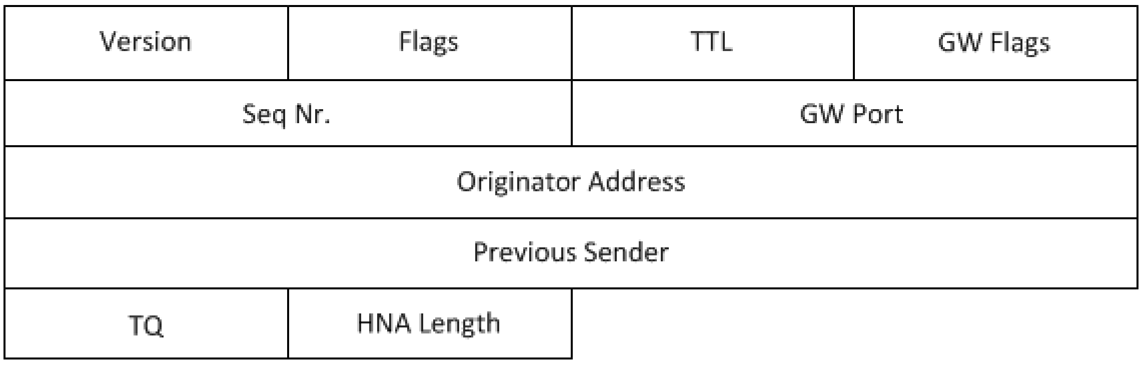


Figure 2.2: BATMAN's Originator Message (OGM) packet format.

the host address of the node 'A' that broadcasted the OGM. When a node 'B' receives this message it checks if the originator address and source address of the IP header are the same - if so the two nodes are direct neighbors. B then forwards the OGM only changing the "Time To Live (TTL)" and "Previous Sender" fields. B's neighbors who receive this OGM from A through B also forwards the packet and so on. All OGMs inside the BATMAN network are broadcasted and rebroadcasted until the TTL has dropped to zero, or until the nodes receive an OGM they have previously sent themselves.

This way all OGMs will be received and rebroadcasted by all nodes in the network and all nodes will learn the existence of each other and which nodes are the first hop between them and the other nodes, i.e. the first hop of the path. All nodes and their first hops in their paths are stored in a list called an "Originator List".

Figure 2.3 shows how an OGM is broadcasted and forwarded throughout the network. The packet originates from the left-most node, and for each node that receives the packet, they forward it to their neighbors again.

When a node which has already received and forwarded an OGM receives the same OGM from another node at a later point - it drops that packet so the network will not get flooded by forwarding the same OGMs until its TTL is zero. This is also necessary in order to prevent routing loops. BATMAN uses sliding windows to detect whether an OGM has been received before or not.

2.2.3 BATMAN Daemon vs. BATMAN Advanced

There are two completely different versions of the BATMAN ad hoc routing protocol[3], and the one described so far has been the BATMAN Daemon, or *batmand*. This version has the benefits of being a network layer protocol making the authentication scheme proposed in this thesis sound. However, there is a new version called BATMAN Advanced, or *batman-adv*, which operates on the link layer instead. This

[3]http://www.open-mesh.org/wiki/open-mesh/BranchesExplained

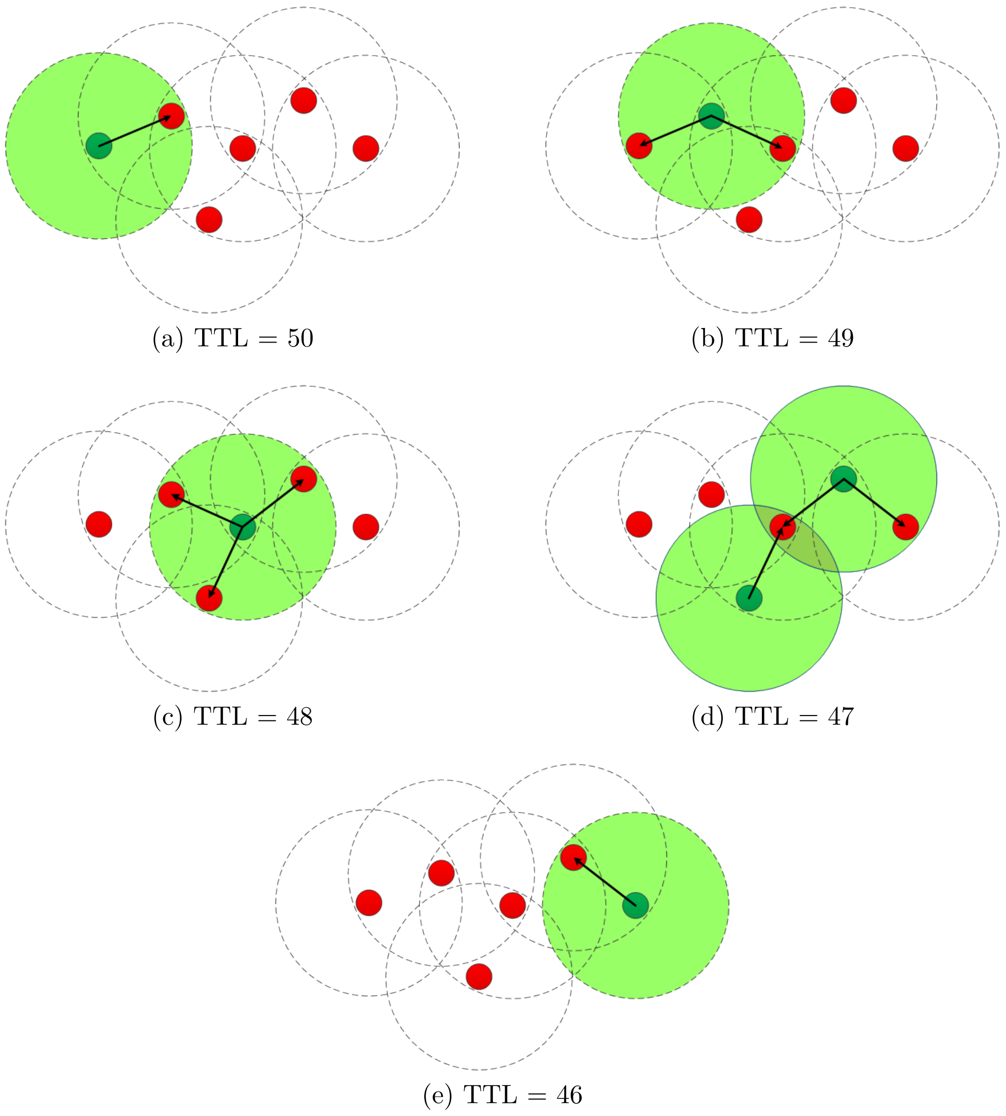


Figure 2.3: Flow of one OGM originating from the left-most node.

version breaks the standard layering principle as it routes packets throughout the network on the link layer, encapsulating everything above such as IP and DHCP packets. Because of this huge difference the design proposed in this thesis will not work with batman-adv, at least not without major changes to the design. Note therefore that whenever BATMAN is mentioned in this thesis, it is the BATMAN Daemon (version 0.3.2) which is being referred to.

2.3 Proxy Certificates

A Proxy Certificate (PC) is a X.509 Certificate "(. . .) derived from, and signed by, a normal X.509 Public Key End Entity Certificate or by another Proxy Certificate for the purpose of providing restricted proxying and delegation within a PKI based authentication system." [TET$^+$04].

The idea behind PCs was to overcome challenges with authentication using e.g. Single Sign-On (SSO) in a Grid Computing setting [FKTT98]. Here a user might want to initiate a process that runs over several entities in the grid, and maybe even after the user has logged off. Therefore a way to delegate the user's rights to the entities running the processes for him was necessary.

Simply understood, a PC is a public key certificate signed not by an Certificate Authority (CA), but by an End-Entity Certificate (EEC) or another PC. With such certificates one can delegate rights on behalf of one self, i.e. if you issue a PC to some other entity, that entity will be able to act on your behalf. PCs allows the use of restrictions, given in a proxyPolicy field. This way the issuer can decide which of its own rights the receiver shall have, granted the issued PC cannot have more or elevated rights than the issuer of the PC.

2.4 Attacks on Mobile Ad Hoc Networks

There are a multitude of attacks on mobile ad hoc networks [GBS10], and a lot of them are essentially Denial-of-Service (DoS) attacks such as jamming. However, in this thesis attacks on the routing model is of more interest and below are two very different attacks on our routing model.

2.4.1 Wormhole Attack

A wormhole attack is an attack in which the attacker is able to set up a so called *wormhole* between two distant nodes in the network, and using this wormhole to e.g. copy a packet (such as in a replay attack) and send it to a receiver at the other end before the original packet arrives (preplay).

Figure 2.4 shows the idea behind the wormhole attack. Here we have a network of trusted nodes (green) and two malicious nodes (red). The trusted nodes are communicating via wireless links, which the malicious nodes are able to pick up. The wormhole between the two malicious nodes can be a fiber channel, or it might just be a wireless radio link where the two malicious nodes have much higher transmitting power than the trusted nodes, extending their transmitting range.

The simplest attack using this model against an ad hoc routing protocol would be for the first malicious node (M1) to copy a routing announcement from node A, send it through the wormhole to M2, and have M2 forward the packet to node B using both mac layer and network layer address spoofing. If successful, i.e. this packet arrives before the re-broadcasts from the other slower nodes, it would fool node B

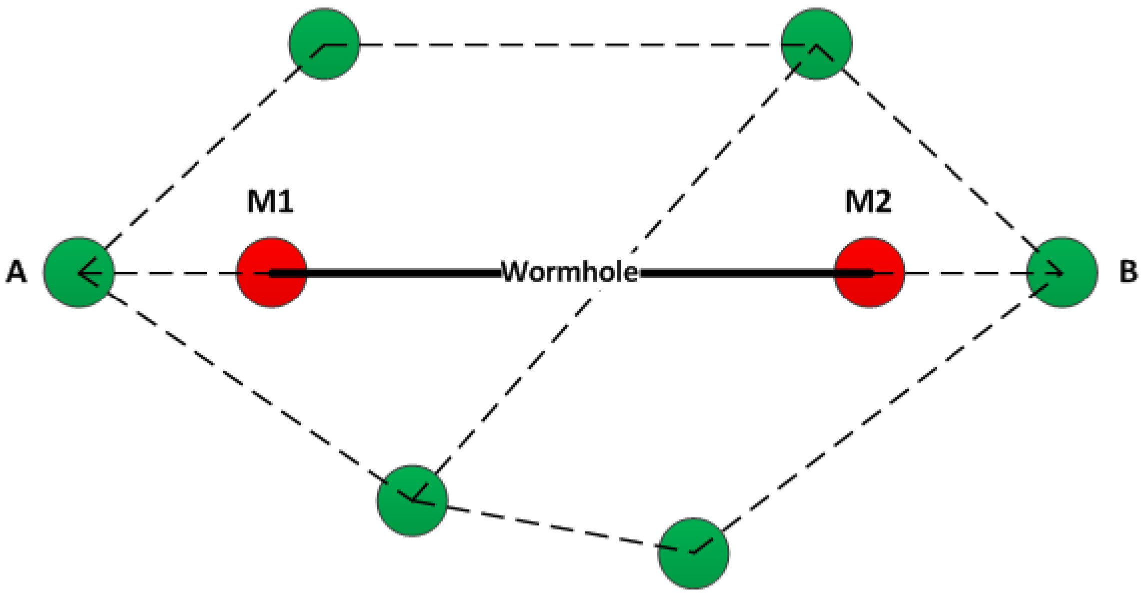


Figure 2.4: Wormhole attack between node A and B.

to believe that node A is a direct neighbor and he would update his routing tables accordingly.

2.4.2 Suppress Replay Attack

A suppress replay attack, sometimes called a pre-play attack, is an attack in which an attacker is able to intercept and suppress a packet, or parts of a packet in air. For instance, if a packet appended with some secret is sent over a radio link, the attacker might be able to jam the first part of the packet and receive the secret. The original recipient would also just receive the secret part, but would on the link layer drop the packet because there would be a Cyclic Redundancy Check (CRC) mismatch (or no CRC at all).

At this point the attacker could be able to create a new falsified packet, and append it with the secret - which the recipient might trust depending on the authentication scheme used.

2.5 Related Work

There are many proposals of security designs for MANETs, and a very few actual implementations based on some specific routing protocols. None of the implementations found by this author would be applicable for the BATMAN protocol, but some of the design proposals probably would.

Seno et al. propose a system design based on a distributed CA, using clustering of nodes and an offline/out-of-band authentication scheme [HSBW11].

Another related authentication scheme is proposed by Dey and Datta which have a more mathematical approach to the problem [DD11]. This solution requires a predefined unique ID and special hardware with a hidden and secure algorithm (i.e. smart card), and is probably better suited for a military application than emergency situations.

Sanzgiri et al. proposed an authentication scheme for ad hoc routing called Authenticated Routing for Ad hoc Networks (ARAN), by adding a signature extension to the AODV reactive routing protocol [SLD+05].

Additionally, some believe the computational costs using asymmetric keys for ad hoc authentication is too great which lead to the Ariadne protocol [HPJ05], and the Secure Routing Protocol [PH02].

Nyre et al. published a design of an hierarchical authentication system for the OLSR ad hoc routing protocol [NJT09].

The problems with most of the related work are their complexity, high computational costs and/or non-scalability when networks running implementations based on their design grow large - and these things are what separates this thesis from the other works.

Chapter 3

System Design

This chapter portrays the design for the proposed implementation of the secure ad hoc network. The design here is based on a functional pro-active ad hoc routing protocol. The routing is left to the chosen routing protocol, i.e. Better Approach To Mobile Ad-hoc Networking (BATMAN), and the changes made will not affect the routing algorithm. The system design is an extension of the protocol, which requires nodes to be authenticated and trusted before being allowed into the network. To strengthen the system each node also has to verify its identity periodically, or it is dropped from the network.

3.1 Brief Overview

In this section a short version of the system design is described. It is far from complete when it comes to explaining the entities, messages being sent, system states, and why certain choices have been made. This section is suggested to read through in its entirety just in order to get an overview of what the system does, leaving the explanations to the subsequent sections.

3.1.1 Initial Authentication

The network setup starts with an out-of-band authentication where a master node, hereafter Service Proxy (SP), verifies new nodes. How this is done can be up to the application, but let us assume that the actors carrying their communication devices, hereafter nodes, physically meet the SP at the scene and verify each others' public keys out-of-band.

When a new node is discovered by the SP using regular routing announcements as part of the pro-active routing protocol, the SP will invite the new node to a handshake to establish a trust relationship between the two nodes. In the invite message the SP shares its certificate. If the node can verify the public key of that certificate, it will request a proxy certificate. If the SP can verify the new node's public key, too, it will issue a proxy certificate with (possibly) the rights to participate in building the Mobile Ad Hoc Network (MANET) by broadcasting its own and re-broadcasting other trusted nodes' routing announcements. Note that the routing announcements of the new node will not have been forwarded by other nodes up until this point, and therefore the new node and the SP would have had to be direct neighbors throughout this handshake.

3.1.2 Continuous Authentication

After being issued with a Proxy Certificate (PC) the newly authenticated node will periodically "broadcast" - unicast to each neighbor - a message containing an ephemeral key and corresponding Initialization Vector (IV), a pseudo-randomly generated nonce, and a digital signature over this message. The ephemeral key is encrypted with the neighbor's public key (hence multiple unicasts instead of an actual broadcast), but the digital signature is generated using the hash of the unencrypted key and the other contents of the message. If you at this point want to see how this keystream generation works, you can jump ahead to Figure 3.6.

After sending this signed "broadcast" to each neighbor, the node itself and its neighbors will generate a keystream from the ephemeral key, IV, and nonce. The node will then append two new bytes from this keystream to each routing announcement, and re-broadcasts of neighbors announcements, sent from this point forward with a sequence number for the recipient to be able to match this "extract" with the sender's keystream at an offset given by the sequence number. If the reader want to see how this message, called a keystream-material message looks like he or she can jump ahead to Figure 3.7.

The neighbors accepts a routing announcement if and only if:

- the routing announcement is appended with a key extract and,
- it matches the sender's keystream at the sequence number offset and,
- that sender's sequence number has not been received earlier (replay).

This "key extract" of the keystream is comparable to regular One-Time Passwords (OTPs) as used by e.g. online banks and it is important to note that they are not "connected" to the routing announcements sent, meaning they do not provide integrity for the packet.

Note also that each node has its own keystream, and that it shares the message above, hereafter “keystream-material message”, with each of its direct neighbors. Each neighbor then use this node’s keystream to verify its routing announcements.

Whenever a routing announcement is forwarded by another trusted node, that node will replace the one-time password with an OTP from its own keystream. This way every node only checks its direct neighbor for authentication, which is a design choice. This proposal assumes that because every node is verified by the SP in the first place, all nodes in the network will be able to trust each other, which also means they will trust their neighbors to properly verify their neighbors again.

When a trusted node discovers a new neighbor, which is a trusted node in the network, they first have to exchange their PCs and verify the signatures on the certificates. Their knowledge of the corresponding private keys to their PCs are verified later when they check each other’s digital signatures on the keystream-material messages. Each of the two nodes then stores the other’s public key, subject name, id, role, and address in a list called Authentication List (AL).

After a neighbor is added to the AL, the node can then also add the neighbor to another list, called a Neighbor List (NL). This list is used to keep track of current direct neighbors and their current keystreams. While nodes in the AL are kept in that list throughout the lifetime of the network, or until the lifetimes of the nodes’ certificates have expired, a node in the NL is removed when it is no longer a direct neighbor.

3.2 Requirements

Ad hoc networks have some desired characteristics such as quick and inexpensive setup and being independent of communication infrastructure, but they also impose great challenges regarding security. The challenges regarding security can vary depending the purpose and environment of the network which will be covered in this section.

3.2.1 Scenario

The design and implementation presented in this thesis is mostly based on an emergency situation scenario, in which a communication infrastructure is unavailable.

If there is a major emergency situation such as an earthquake or tsunami, it is likely that parts of or the entire communication infrastructure at the scene is destroyed or temporarily down. The remaining communication lines will probably be congested, so only a small amount of the communication actually goes through.

In this situation, it is of great importance that Emergency Personnel, such as Paramedics, Firemen, Policemen and possibly even the Military, are able to communicate efficiently and therefore independently of the public communication infrastructure. They need a network in order to manage the the operation, and therefore availability is probably the most important trait of this network. Secondly, they should be able to trust the communication on the network - i.e. messages sent are from whom they claim they to be.

Also, being able to authorize new actors on the scene, such as the Red Cross, can be critical to the operation. These new actors will probably not have the necessary authentication tokens, i.e. certificates, required by the authentication scheme in the network.

3.2.2 List of Requirements

Based on the scenario above these requirements can be extracted and made into general requirements that needs to be addressed by the system design. The work presented here is based on several sources, most prevalent being the research from the OASIS project [Sva08] [TJN09] [NJT09] and the doctoral project of Eli Winjum carried out at UniK [WSK06].

#	**Requirement Description**
R1	A node must be authorized in order to get full rights in a network [DLRS01], [SDL+02]
R2	A node without a recognized authentication token should be able to become authorized if necessary
R3	Networks need a master node to handle authentication of new nodes
R4	Access control (after initial authentication) should work without centralized nodes
R5	Different networks should be able to collaborate [WSK06]
R6	Only master nodes can decide access policies of users/nodes
R7	Nodes must not be able to alter their access policies

Table 3.1: Requirements based upon our simplified and general scenario.

An early study produced security requirements of ad hoc networks demanding that the routing logic must not be spoofed or altered to produce different behavior [DLRS01]. R1 is constructed from that requirement. During the OASIS project, a requirement ensuring different actors such as police, fire and medical professionals can participate in the network, gives R2 [TJN09].

Because of R2 there needs to be some sort of authority managing the authentication and access management, which leads to R3. However, verifying nodes access rights after the fact should be possible even without the availability of central managing nodes, SPs, (R4) - also a requirement given by the OASIS project.

The doctoral project of Winjum recommends seamless radio coverage over the whole crisis area, possibly requiring merging or at least collaboration between different networks, R5.

R7 comes implicitly from R6 because R6 would be useless if regular nodes could alter their privileges without the permission of a master or management node. R6 is necessary in this design as no authentication is required prior to the network setup, and it is therefore no way to know which rights one actor/node should have. As will be discussed in Chapter 6 the network might also be able to recognize authentication tokens, such as long lived certificates, issued prior to this setup. If this is the case, one might have to re-evaluate or account for this in these requirements.

The OASIS project had another important requirement which is not covered here, but is important to mention - there should be mechanisms in place to detect misbehaving nodes, i.e. already trusted nodes that act maliciously. This detection is not covered in this thesis as pointed out in 1.4.2, but is nevertheless important to take notice of.

3.3 Why use Proxy Certificates?

PCs, as described in Section 2.3, are used to delegate rights on behalf of their issuers. That means that the issuer, i.e. the SP, can choose to delegate all or a subset of its rights to the receiver of the PC. This can be very useful in a situation where the nodes themselves are unable to properly authenticate themselves with their pre-existing Long-Lived Public Key Certificates (LLPKCs) because the SP on the scene has no way to verify their certificates. This can be true if their certificates are issued by an unknown root certificate (Certificate Authority (CA)) or simply if there is no Internet access and the certificate is signed by an unknown entity.

In addition, proxy certificates commonly have a short valid lifetime compared to regular certificates, meaning an implementation using proxy certificates does not necessarily need to implement a certificate revocation scheme, which makes for less management operations in the management-hostile environment that are MANETs. If a certificate is compromised in a MANET, the time of exposure is limited because of the short valid lifetimes on the PCs. Regular LLPKCs on the other hand, could be compromised throughout the lifetime of the network, or until revocation lists were brought to the scene, out-of-band or by achieving Internet access later on.

Also, the SP could be interested in giving the node rights the node would not usually have, depending on the situation. This is easier to achieve when the SP can delegate its own rights. Different nodes can be given different rights, as long as they are a subset of the SP's rights. There are countless of different potential rights that can be useful for a network, given the situation they are used in, and here is a few possible rights/privileges to give the reader an understanding of the possibilities they give:

- Announce itself - let the MANET know of your existence
- Re-broadcast other nodes' announcements - reshape the network topology
- Announce a gateway - give the MANET access to another network or the Internet
- Use the gateway - allow you to communicate outside the MANET
- Send and receive messages with a defined application - full application rights
- Only receive messages from a defined application - limited application rights

The different choices are essentially up to the SP managing the network. One can ask why this is necessary, and again it depends on the application. If you are setting up a MANET on the scene of a disaster to assist emergency personnel, you could have some actors be able to organize the effort by sending orders/commands to the other actors, while some actors only are allowed to receive the orders. In this situation it might be of great importance to know that only verified nodes are able to give commands, but the importance of getting this information available outweighs the need to verify the nodes/actors receiving this information.

3.4 Design Overview

The secure ad hoc network designed here does not change any fundamental workings of regular ad hoc routing protocols. When nodes have been authenticated and they have verified their neighbors respectively, the routing announcements are generated, broadcasted, forwarded, and handled by the routing protocol almost as usual. The only addition to the routing protocol is the addition of one-time passwords (OTP) to the routing announcements, and the handling of said OTPs. The routing algorithms themselves are left completely unchanged.

The proposed design should work with most pro-active ad hoc routing protocols operating on the network layer with limited alterations - but this design is specifically made for the BATMAN [NALW10] routing protocol chosen for its simpler design compared to e.g. OLSR [CJ10] and because it operates on the third layer of the Open Systems Interconnection (OSI) model [Zim80]. Whether this design would work on a link-layer protocol is unknown, and there is still a discussion whether having routing protocols on the link-layer is a good thing, as it breaks the layering principles of the OSI model [MDK10]. How this design is incorporated, or added, to the BATMAN protocol will be explained in Chapter 4.

The basic principle of the proposed design is that an authenticated node accepts other authenticated nodes' routing announcements and forwards them as normal, while discarding routing announcements from unauthenticated nodes. One or more nodes in the network will assume a role as master node(s), or a SP, with the extra capability of authorizing new nodes into the network. A special certificate called a PC [TET$^+$04] will be used for authentication after this authorization has taken place such that other nodes in the network will be able to authenticate and accept the new node.

3.4.1 Entity Explanation

Before a simplified example can be given, a few new entities in this design needs to be explained further. This is the short version, just enough for the reader to understand the example - the full description of these entities and why they are necessary will be given in Section 3.7. Some of these entities are portrayed in Figure 3.1. The portrayed entities will be used as a template for other figures later in this thesis report.

- **Service Proxy (SP)** is responsible for tasks similar to that of a CA and has the master role in the network. The SP is the entity that authorizes new nodes and signs their PCs.
- **Proxy Certificate 0 (PC0)** is a PC belonging to a SP signed by its regular LLPKC. This PC has a certificate depth of 0, thus we refer to it as a PC0.
- **Proxy Certificate 1 (PC1)** is a PC signed by a PC0 (i.e. by the private key of the SP). All authenticated nodes in one network, have at least one PC1 signed by a SP from that network.
- **Authentication List (AL)** is a list containing the necessary information about all known and authorized nodes in the network. All nodes keep a local copy of the AL which they use to authenticate other nodes in the network.
- **Neighbor List (NL)** is a list containing the current trusted direct neighbors with a copy of their keystreams used for verifying their routing announcements. The NL is a subset of the AL and a node must be found in the AL before it can be added to the NL.
- **Authenticated/Trusted Node** is a node which has been issued a PC1 from the SP and is considered a trusted node in the network. This node can take part in sending its own routing announcements and forwarding other authenticated nodes' routing announcements, i.e. they take part in changing the network topology.
- **Unauthenticated Node** is a node which has not yet been authorized, or has been denied access by the SP. It does not possess a certificate for which the other nodes can verify, and its routing announcements are ignored by other trusted nodes in the network.

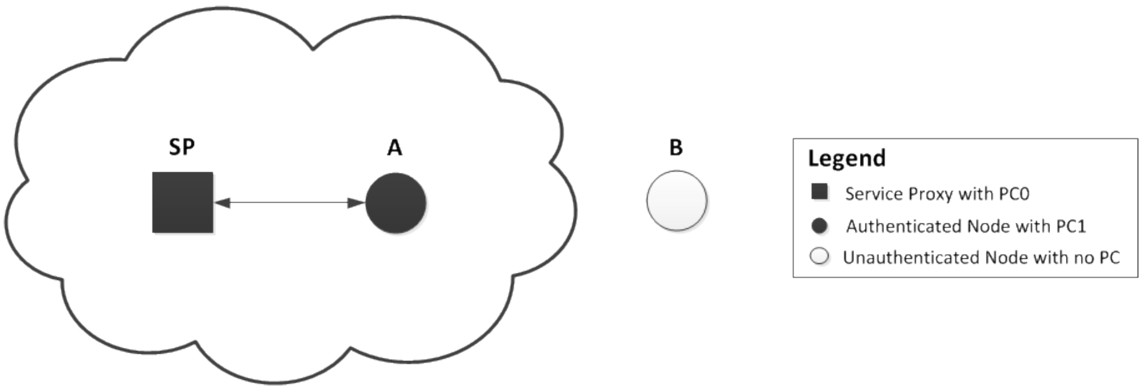

SP
A
B
Legend
■ Service Proxy with PC0
● Authenticated Node with PC1
○ Unauthenticated Node with no PC

Figure 3.1: Different entities in the Simple Example.

3.4.2 Simple Example

Two nodes are within transmitting range of each other, i.e. they are direct neighbors. One of the nodes is a SP and the other is unauthenticated. The pro-active ad hoc routing protocol used on both nodes regularly broadcasts routing announcements, so the two nodes learn of each others' existence - i.e. they "discover" each other. Upon reception of a routing announcement from the unauthenticated node, the SP will invite the node for a handshake. The invite message contains the SP's PC0 which assumably the unauthenticated node is able to verify, possibly based on a prior out-of-band sharing of public key fingerprints.

After verifying the PC0, the unauthenticated node will send a PC request with its own public key. If the SP is able to verify the sender's public key (same assumption as above) and the SP decides this node should have access to network, it will create and sign a PC for this node - i.e. the node is issued a PC1.

When the handshake completes both nodes will add the other node of the handshake to their AL - storing their id, address, unique subject name, role, and public key. All the steps up to this point is portrayed in the first half of Figure 3.2. As illustrated by the color of the node circle, the new node (A in the figure) is authenticated after receiving the issued PC1.

Next, both the newly authenticated/trusted node and the SP will send one another a packet called a "keystream-material message" containing their current (or new in the case of the trusted node) ephemeral key, IV, nonce value and a digital signature over the hash of these values. Before sending this message, the ephemeral key part is encrypted with the recipient's public key to keep this information secret. Note that the digital signature however, is computed over the unencrypted key in order to re-use this signature for all neighbors the node has to send this message to.

Both nodes can now generate each other's keystream, hence the name "keystream-material message", used to verify the sender of subsequent routing announcements. The keystream, address, id, an empty "last sequence number", and an empty sliding window of the other node is then stored in the NL.

From this point forward, the two nodes use a two-byte extract of the keystream as an one-time password and appends it, together with a sequence number (for finding the correct offset in the keystream), to future routing announcements. This goes for both original routing announcements from the node itself, and when they forward other trusted nodes' routing announcements. The two nodes never re-use the same extract from the keystream, hence the name "One-Time Password (OTP)", and they use a sliding window stored in their NL to keep track of which extracts they have received from their neighbor. This last part is crucial in order to be able to drop announcements containing re-used extracts, avoiding replay attacks.

Whenever a new node is discovered by the SP the procedure above repeats, and a new addition is made to the AL and NL. Other previously trusted nodes will learn the identity of new authenticated nodes when they discover the new node and initiate a PC and keystream-material exchange, which will be discussed later.

Figure 3.2 shows a message sequence chart of the messages sent between two nodes during the simple example. The second half portrays the messages sent in a keystream-material message and that they are added to the NL after these have been received, and after the AL-addition. Routing announcements without One-Time Passwords are sent periodically throughout the sequence until keystreams are generated and shared, but this is left out of the figure as they do not affect the nodes. Only the routing announcements in the beginning and in the end that are appended with OTPs are portrayed as they show how the handshake is initiated (by discovery) and ended after successfully ending the handshake and keystream material sharing.

If a new node enters transmitting range of the two nodes, similar messages are exchanged, as shown in Figure 3.3.

Here we see that after node B has been fully authenticated and started broadcasting routing announcements appended with its own OTPs, he and node A will "discover" each other. Up until this point node A has ignored node B's routing announcements because they have not been appended with any valid/recognizable OTPs.

After they've discovered each other they share their certificates as seen in the figure. Before continuing they need to verify each other's certificate, which is to verify the signature on the PC1's up against the public key of the SP.

When both nodes have verified each other's certificates, they send each other their keystream-material messages, in which they verify each others' signatures to actually verify each other as the legitimate owners of the certificates. If they are able to verify each others' signature they will add each other to their AL and NL, respectively.

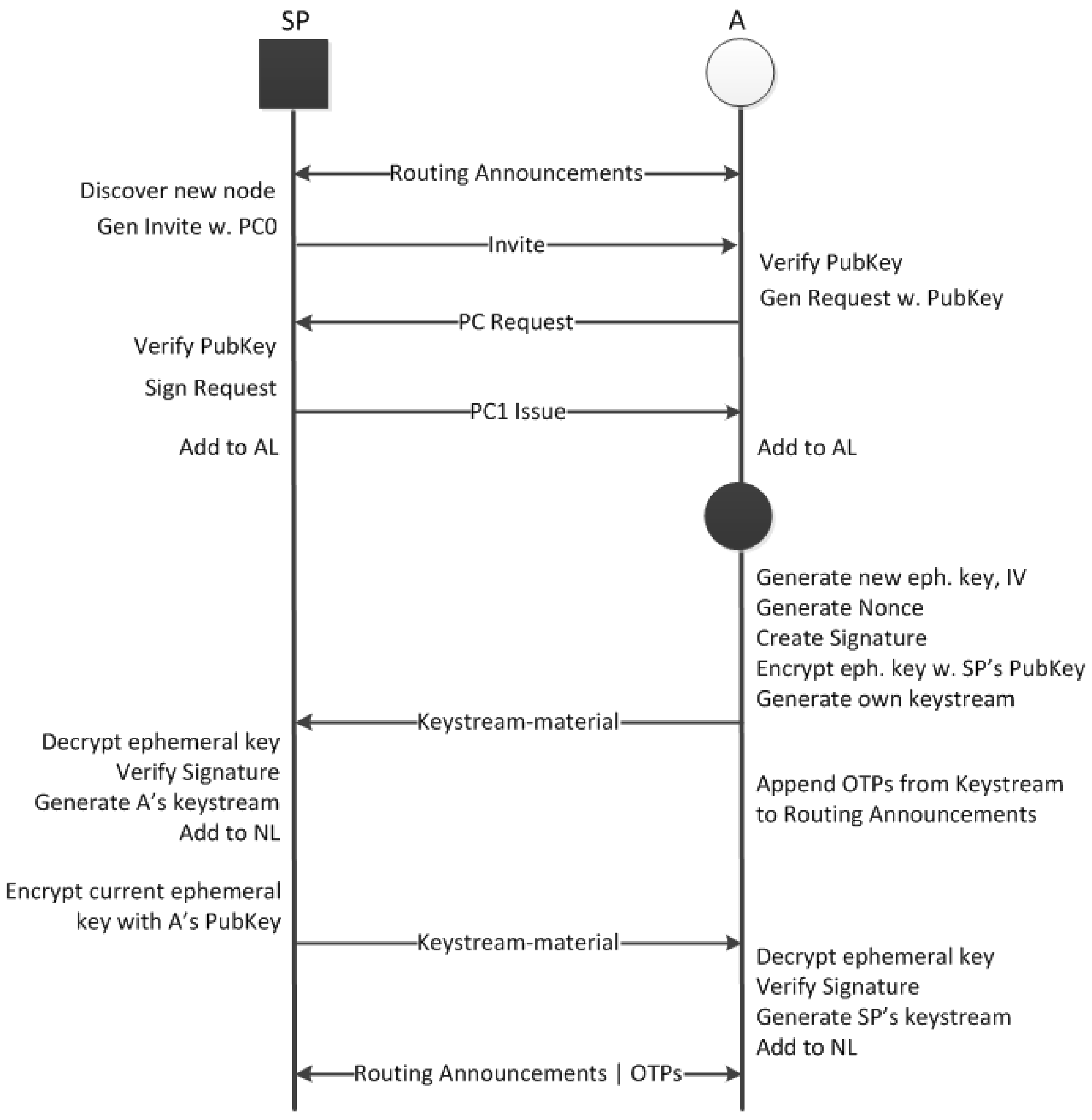


Figure 3.2: Authentication handshake and keystream-material sharing between a new node A and the SP.

3.5 Authentication Phase

This section is devoted to explain the phases a node goes through before and when authenticating itself to the network. These phases should be similar for most network layer pro-active routing protocols as the main trigger of these phases are the routing announcements sent as per normal operation of any pro-active routing protocol. This phase is necessary due to requirement R1 from Table 3.1.

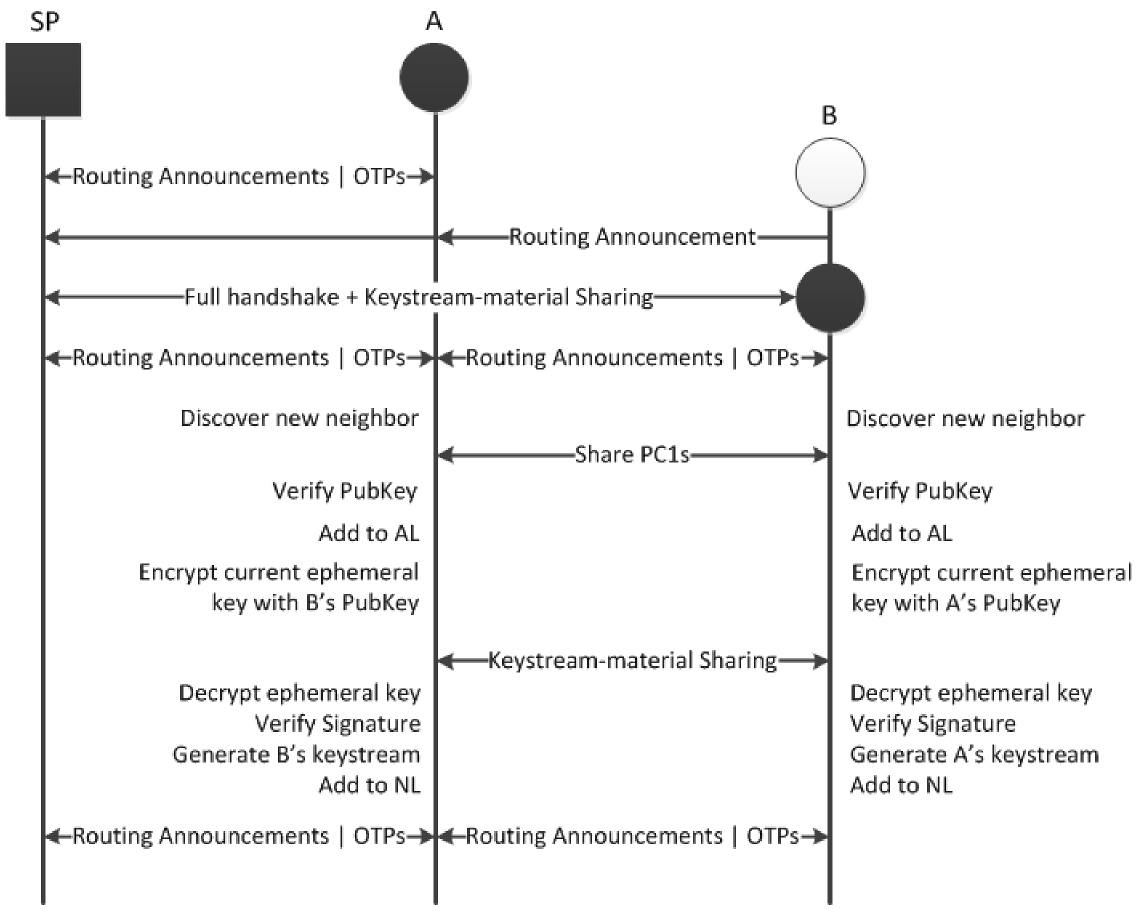

SP
A
B
Routing Announcements | OTPs
Routing Announcement
Full handshake + Keystream-material Sharing
Routing Announcements | OTPs
Routing Announcements | OTPs
Discover new neighbor
Discover new neighbor
Share PC1s
Verify PubKey
Verify PubKey
Add to AL
Add to AL
Encrypt current ephemeral key with B's PubKey
Encrypt current ephemeral key with A's PubKey
Keystream-material Sharing
Decrypt ephemeral key
Verify Signature
Generate B's keystream
Add to NL
Decrypt ephemeral key
Verify Signature
Generate A's keystream
Add to NL
Routing Announcements | OTPs
Routing Announcements | OTPs

Figure 3.3: Another node B joins the network from previous figure.

3.5.1 Node Discovery

Upon entering the network area, the node is both unauthenticated and unknown to the network. Because it uses a pro-active routing protocol, the node regularly broadcasts routing announcements to be received by any potential node in the area. At this point an assumption that all nodes are configured with unique addresses and with the same netmask is done in order to focus on the security challenges, rather the non-trivial task of assigning addresses in MANETs (See Limitations in Section 1.4.1). With this assumption all nodes within transmitting range of the new node can receive its broadcasts.

Simultaneously, the node also listens to other nodes' routing announcements. Depending on the time interval between the broadcasts and whether the nodes within each other's transmitting range are asymmetrical, they will discover each other approximately at the same time.

Figure 3.4 illustrates the routing announcements periodically sent by two nodes until they discover each other, and how the SP enters the authentication handshake state while the new node does nothing. One of the nodes have already assumed the

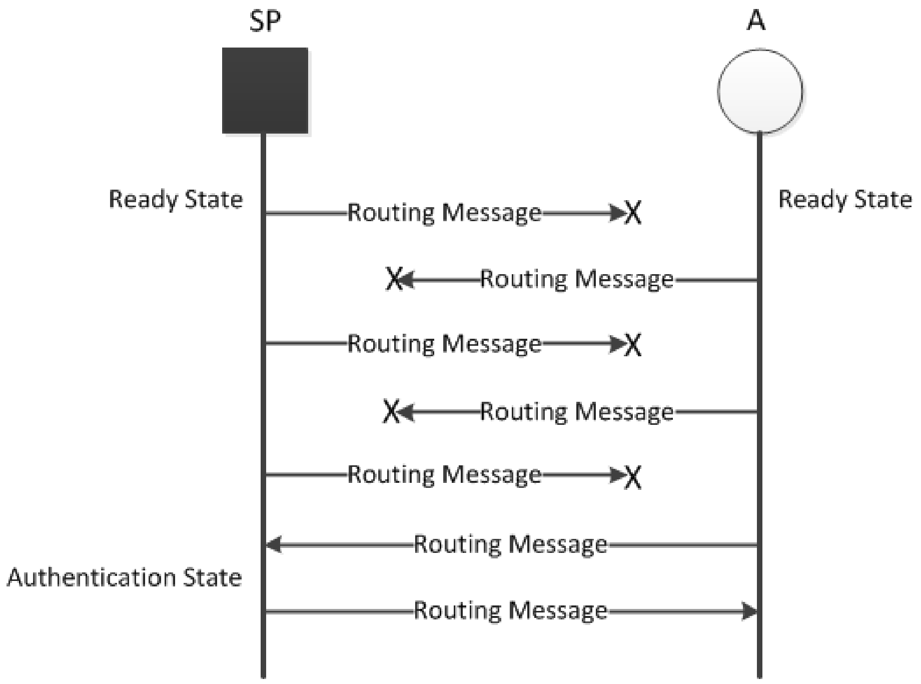


Figure 3.4: Discovery Phase between a SP and an unauthenticated node A.

master role and is a SP while the other node is unauthenticated.

The SP will have a PC0 and its AL has only one entry - itself. Note that if it had authorized another node at an earlier point in time (but within the lifetime of the PC) that node's values would also be represented in the AL, even if the node was outside the network at this point in time (physically).

The new node does not have any PC at this point, unless it has a PC issued within and valid only for another network. This is however not covered here, and it is assumed the node has no certificate at all. The same goes for its AL, or one can rather say it has an empty AL.

3.5.2 Authentication Handshake

Once the two nodes have discovered each other, the SP will enter an authentication handshake state, while the unauthenticated node will not do anything. The authentication handshake state of the SP, and later of the other node, does not obstruct regular routing announcements to be sent. The current state only affects how the Authentication Module (AM) operates, not how the original routing protocol operates. Actually, the handshake and all other messages and operations handled by the AM is executed in a separate thread and sent and received using another socket than the rest of the protocol. This is further elaborated in the next chapter.

Because the unauthenticated node does not enter a new state upon the discovery

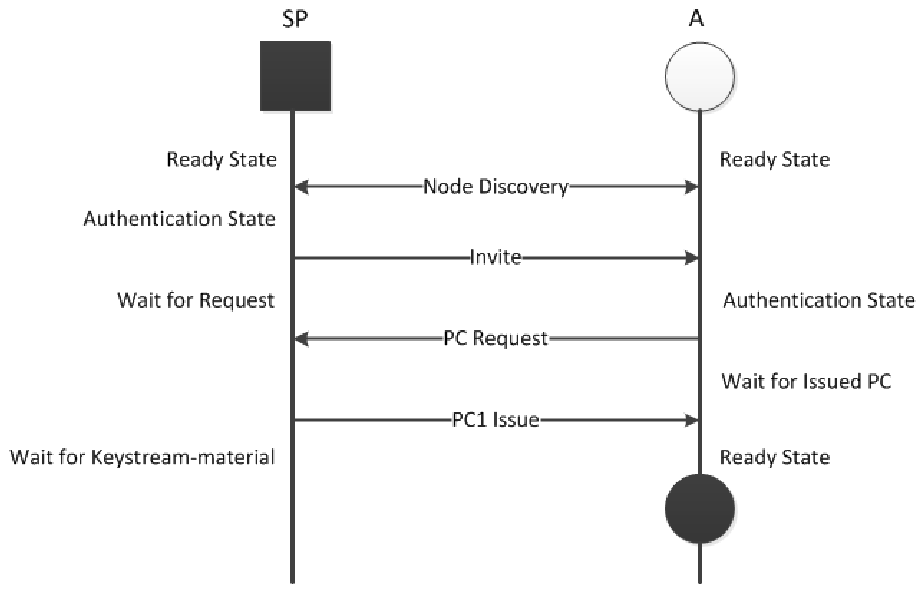


Figure 3.5: Handshake between a SP and an unauthenticated node A.

of the SP, which could just be a regular authenticated node as well, there will be no deadlock if for some reason the SP should never initiate, or invite to, the handshake. This could happen for a multitude of reasons where the two nodes lose their connectivity between each other because of the flaky nature of wireless ad hoc networks, or the assumed SP was only a regular authenticated, trusted, node. From the routing announcements it is not possible to derive whether a node is simply authenticated, or if it also has master role capabilities being a SP. It is only possible to derive, or rather it seems, that a node is authenticated in some network or not.

While the new node is "waiting" for an invite, or simply not doing anything other than announcing its existence - the SP generates an invite message which is a message containing its PC0. The invite will be directly addressed to the new node, and not broadcasted as regular routing announcements are. The SP will then wait for a certificate request (PC Request) for a short predefined time before aborting the handshake and entering its ready-state. If the two nodes discover each other once more, the SP will once again try to invite the other node to the handshake. If this fails several times, the SP will eventually ignore all the discoveries of the other node (based on address) for a predefined time before it tries again.

If the new node is able to verify the SP's public key it will use its public key pair and generate a PC request which it will send back to the SP. This request abides by the rules for making a proxy certificate [TET$^+$04], setting the "Issuer Name" the same as the "Subject Name" from the received PC0, the "Subject Name" as the "Issuer Name" appended with its own unique Common Name, which is the hash value of its own public key, and setting the "Serial Number" to the same hash value.

Before issuing the PC1 the SP also has to verify the public key received in the request message. As before, the knowledge necessary to be able to verify the public key is assumed to have been communicated out-of-band prior to this setup. The PC1 is appended with the proxy policies the SP deems fit for the node, and then signed with the SP's private key. After sending the PC1 to the new node the SP will add the new node to its local AL and the new node will, after verifying the signature on its newly issued certificate, store the issued certificate for later and add the SP to its own local AL. In Figure 3.5 the different states associated with the different messages during the handshake is shown.

3.5.3 Out-Of-Band Authentication

Above, only brief mention was made to how the initial verification of each node's public keys were made. For this purpose, many authentication schemes have been produced, but when it comes to MANETs, possibly with no Internet connection, proper authentication becomes a difficult task. In the discussion in Chapter 6 different schemes will be discussed, but here only one "scheme" is accounted for.

If you have no pre-shared information between the parties involved in the network, the simplest way to authenticate a new node is to use an out-of-band authentication. This adheres to the R2 requirement in Table 3.2. The implementation of such an authentication scheme will be discussed in the next chapter, and will only be briefly mentioned here. In the PGP model new users will have to share their public key fingerprint with the SP physically or through a different communication channel than the one the authentication process is supposed to secure, and vice versa [Zim95]. By doing this, the SP can store the fingerprint and use it to check the received public key in the PC request for authenticity in order to make sure the new node is run by the actual person he met and verified physically. The new node can similarly verify the SP.

To implement this, the application running the routing protocol and the authentication service could either read a file for "allowed" fingerprints, or simply have user interaction with a pop-up window showing the fingerprint and asking the user whether to trust the public key or not.

In this design chapter and in the implementation chapter this part will be ignored, and rather assume that if you are a direct neighbor of the SP, you are automatically allowed to enter the network and therefore no real verification of the certificates are done. This must however, be thought through and most likely changed (!) in a real-world implementation.

3.6 Authorized Operation

When a node has been issued a PC and become a trusted node in the network, it is almost ready to take part in the sending and forwarding of routing announcements.

But before a node can take part in the routing in the network, there has to be a way for the node to continuously re-authenticate itself to make sure the node is still who it says it is.

A common practice would be to sign each and every routing announcement from this point forward, and this would probably work in terms of authentication. However, as the network become "flooded" by routing announcement broadcasts, these signatures simply makes the packets too large. A signature from a 1024 bits RSA key can be up to 1024 (maximum) bits in raw size, or approximately 1392 bits long if encoded with Base64 encoding. In comparison the whole routing announcement packet used in BATMAN is only 144 bits.

As seen, signing every message just does not scale very well when signatures can be close to 10 times the size of the original packet. The first solution to this problem many might think of would be to only sign a very few of the announcements, periodically. This however, would be totally disastrous as this would have no protection against spoofing attacks whatsoever. An attacker could wait for a legitimate node to send a signed announcement and after this send his own fake announcements spoofed with the legitimate node's addresses.

The novel solution proposed here however, takes use of OTPs instead. The newly authenticated/trusted node, talked about earlier, now has to be able to append **OTS!**s (**OTS!**s) to its routing announcements, and to verify other nodes' OTP as it receives routing announcements from them.

3.6.1 Keystream generation

The OTPs are actually smaller extracts of larger keystreams shared between direct neighbors in the network. When a node started its routing protocol daemon and before discovering the SP in the previous step, it generates a high entropy pseudo-random master key and a *regular* pseudo-random IV. The master key needs high entropy to be as random as possible, because the security of this design relies on this key to both be secret and not guessable.

When the node is ready to share its keystream for the first time, the node will generate a new ephemeral key by encrypting $K_{ephemeral} = E_{K_{master}}\{i\}$ where $i = 1, 2, 3 \ldots$ and a corresponding IV generated in the same manner as the previous IV for the master key. In addition, a large nonce is generated with a pseudo-random function, also in the same pseudo-random fashion as the IVs. To elaborate, this pseudo-randomness does not need strong randomness with a high entropy like the master key, it only needs to be (with high probability) different each time it is generated.

With the current ephemeral key, generated with $i = 1$, IV and nonce, the node can now generate its first keystream to be used as OTPs for its routing announcements.

The keystream is generated by encrypting the nonce with the ephemeral key multiple times, each time extending the size of the keystream. Each encryption is a full "Cipher-block chaining" AES encryption, i.e. each encryption step referred to here (and later) are actually multiple encryptions using the AES-CBC mode.

In the first encryption (full AES-CBC) the supplied IV is used, while in the next iterations a 16 byte extract of the output from the previous encryption (ciphertext) are used as IV. To explain, this is basically a Cipher-block Chaining (CBC) mode encryption of multiple AES-CBC blocks. Figure 3.6 shows the process assuming the ephemeral key, IV and nonce have already been created.

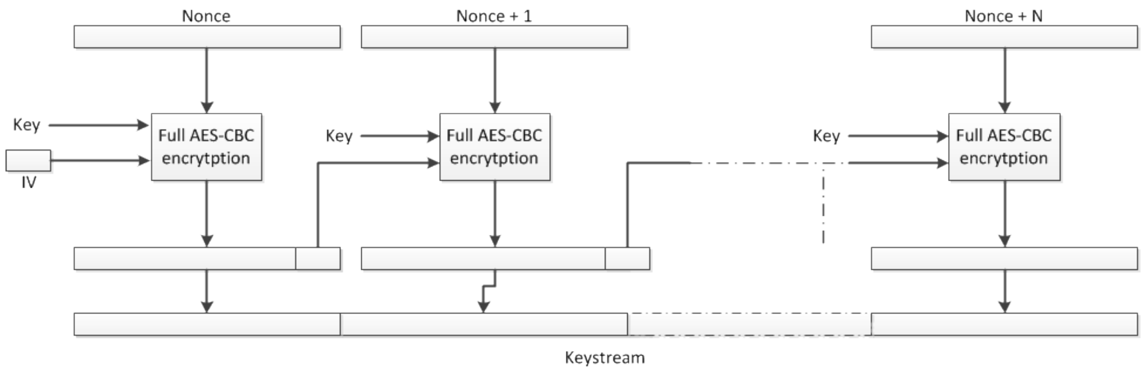


Figure 3.6: Keystream generation based on the supplied Nonce, Key and IV.

In the figure there are some boxes marked with "Full AES-CBC encryption". This is meant to illustrate that within each of those boxes there is a full AES-CBC encryption taking place, with multiple steps according to the OpenSSL implementation of AES-CBC. The reader should also notice that as the ciphertext output of each of those boxes are much larger than the AES block size, therefore only an AES block size extract at the end of the ciphertext is used as an IV for the next encryption step, instead of the whole ciphertext which would usually be done in CBC as there the ciphertext corresponds to the correct block size.

The lifetime of a node's keystream is determined by two factors. First, the keystream is updated at a regular time interval (60 seconds is used in this thesis' implementation). Second, the keystream could be exhausted before this time interval. For whichever comes first, a new keystream has to be generated, and shared with all direct neighbors. Each time a new keystream-material message is created, the ephemeral key is generated by increasing the plaintext i. I.e. the second time a keystream is generated, the ephemeral key is $E_{K_{master}}\{2\}$. Note that for the new keystream, a new IV and a new nonce are also pseudo-randomly generated.

Now, before this keystream can be used to authenticate routing announcements, the node will have to share the keystream with its direct neighbors. This is done by sending all the material necessary for the neighbor to create the keystream themselves and signing the message with your private key. I.e. the node will have to send the ephemeral key, IV and nonce to its neighbor. Here it is crucial that the key stays secret between the neighbors, therefore the node will have to encrypt the ephemeral key with its neighbors' public keys and send a unicast message with the keystream material to each one of them. The details around sharing the keystreams will be further explained in the following section regarding all messages sent in the network (3.8).

The design behind the keystream changed during implementation, and is now a hybrid between different known keystream generation schemes. This scheme would possibly suffice, but should be replaced with a well known and tested scheme before put to use in a real life scenario, or put to a great deal of testing before use. A well known scheme that could be used is the S/Key scheme [Hal94] where if N amounts of OTPs are needed, they are generated by creating Hash^1(secret), Hash^2(secret), . . . , Hash^n(secret), where each hash or digest value is one password. The Hash^n(secret) is sent to the neighbors in the keystream-material sharing message while the passwords are stored in the reversed order (from n to 1) in the senders keystream. The secret is discarded.

3.6.2 Using One-Time Passwords from Keystream

Once the keystream has been shared with the neighbors, the node will begin to extract smaller chunks of the keystream to use as OTPs for each routing announcement it generates itself, or forwards from other trusted neighbors. In addition to the OTPs appended to the announcements, the node keeps a counter, or sequence number, to keep track of which OTP it has used in order not to use the password twice.

Similarly the node will have to both verify the correctness of the OTPs received in announcements from its trusted neighbors, and that they have not been received before. If the OTP has been received before the announcement needs to be dropped, and assumed to be part of a replay-attack. If the node would have accepted re-using of one-time-passwords, an attacker could listen and record valid one-time-passwords and re-use them with false routing information in order to disrupt the routing in the network.

How to check for replayed one-time-passwords is further elaborated in the next chapter, but simply put - a sliding window that records the last announcements is used, setting a bit value to true if an announcement with the corresponding sequence number has been received, and zero if not.

Note that this scheme is fully based on trust. You trust that your trusted neighbors only send you their own announcements and forwards announcements from its trusted neighbors. One can therefore say there is a trust-chain where you do not explicitly verify the authenticity of all nodes' announcements, because you trust your neighbors to verify it for you, and their neighbors to verify it for them again and so on. This scheme will not detect malicious behavior if some trusted node in the network is compromised, and would allow the malicious behavior to continue. This is however mentioned under limitations in Section 1.4.2.

3.6.3 Discovering Additional New Neighbors

Up until this point, the node has only discovered the SP and after the authentication handshake shared their keystream material in order to be able to trust each others' routing announcements. Because of the trust mode being used in this design, each routing announcement from the SP are trusted, even if they are re-broadcasts (forwarded by the SP) originating from other nodes not yet known to the new node. This means that the routing table of our node could possibly fill up with unknown nodes, reachable through SP. To have all data streams go through the SP however, would be less than ideal. If some of those nodes are direct neighbors of our node, they should be able to communicate directly.

Therefore a discovery mode of other trusted (by the network at least) nodes needs to be handled. Until now, if you received any routing announcements from an unknown neighbor you would have dropped the packet. After becoming an authenticated node, the node will now instead engage in a exchange of PCs with the other neighbor, given the neighbor is authenticated. If the neighbor sends you a PC with a signature of the SP which you are able to verify, you will add the node to your AL and send him your PC - if you have not sent it already.

The verification of the ownership of the PC is not performed before receiving the keystream-material message which occurs soon after sharing PCs. As a remainder the keystream-material message is signed with the private key of the sender. Therefore, assuming the nodes has not been compromised, the ownership of the neighbors are verified by checking the validity of said signatures.

3.7 Detailed Entity Description

3.7.1 Proxy Certificate

In a scenario where actors try to communicate with each other, without the Internet or other communication infrastructure available one cannot depend on a Public Key Infrastructure (PKI) for authentication of the actors. One needs the option of being able to verify actors physically (out-of-band) and then issuing them an authentication token to be used during this scenario. The reasoning to use a certificate at all is due to requirement R4 from Table 3.1, because with an easily verifiable certificate for all trusted nodes in the network all nodes can continue to verify each other even if the SP is unavailable. This is where the Proxy Certificates comes in handy, while regular End-Entity Certificates (EECs) are not as easy to verify, and also should not be used for issuing new regular certificates, per RFC2459 [HFPS99].

Using PCs this issue can be handled in a well-defined manner. The field "pCPath-LenConstraint" is used for constraining the depth of the certificate chain below the PC itself. I.e. if a PC has a pCPathLenConstraint of zero, the PC cannot issue any other PCs itself, while if the pCPathLenConstraint value is one the PC can be used to issue new PCs which again will have a pCPathLenConstraint of zero, and so on.

For our scenario, the pCPathLenConstraint value should never exceed one, or else it will once again be complex to handle path validations. Every node trying to verify a PC need to know the issuer of the certificate directly, and not indirectly through root CAs as is possible if you have a PKI.

A PC is almost a regular certificate, except for some values that are forbidden to use, some values that are required to use, and some values that must be used in a well defined way [TET+04]. Below are the special fields that are included in this system design:

- Subject Name: <Issuer Name> + "CN: <SHA-1(Public Key)>"
- Issuer Name: <Subject Name of Issuer>
- Serial Number: "<SHA-1(Public Key)>"
- X509v3 Extensions:
  - Key Usage: "Digital signature, Key encipherment"
  - ProxyCertInfo:
    - pCPathLenConstraint: <0 or 1>
    - proxyPolicy:
      - policyLanguage: "id-ppl-anyLanguage"
      - policy: "Role:<sp or authenticated>,Routing:<full or limited>, Application:<full or limited>"

Per RFC3820 [TET+04] the subject name has the constraint that it "should be unique amongst all Proxy Certificates issued by a particular Proxy Issuer". The modal verb "should" used in this requirement is used in order to ease this requirement so that a method creating a unique name with a high probability, but not provable uniqueness, may be accepted. Therefore using a SHA-1 digest of the public key used by the node is regarded as "good enough", or "unique enough". The reason why this subject name should be unique is because it is the identity which we later try to verify during authentication.

The same uniqueness requirement is true for the serial number, and allows the serial number to be the same as the subject name except for the "CN:" prefix.

The specifications requires the "Digital signatures" value for the "Key Usage" extension because a PC can be used to sign the keystream material messages. "Key encipherment" is also necessary in this extension because other neighbors need to use your public key to encrypt their ephemeral keys when they want to share their keystream with you.

The "policyLanguage" field is set to the default value, leaving the language used in the policy field later to be handled correctly by the application, rather than being universally understandable.

Finally, the last field called "policy" might be of greatest interest. This field explains what rights the given PC has for this design. Three values have been defined for use in this design:

- Role - Node's role in the network
- Routing - Whether the node can partake in the routing
- Application - Whether the node has access to the application layer

These values are specific to this design, and might not be desired in other applications. The first value, "role", is used to declare which role the node has been given. It can be "sp" which means the node as administrative rights in the network, or "authenticated" meaning the node is trusted by the SP and should be trusted by all other nodes in the network.

The "routing" value can be "full" or "limited". If it is full the node is able to generate and broadcast its own routing announcements, and forward its trusted nodes' routing announcements. This means the node partakes completely in the network and might be placed in a path between two nodes in the network. The "limited" value is used so a node can become an end-node, but not a node in a path between other nodes. I.e., it can send its own routing announcements, but it cannot forward other nodes' routing announcements. If a node with limited rights does forward other nodes' announcements they must be dropped. In addition, a detection system of misbehaving nodes should pick up this as potentially malicious behavior, but this is not being implemented as previously explained in Limitations (1.4.2).

The last value, "application", is used to declare whether the node should have access to the upper layers and being able to use the applications running on top of the MANET. This restriction is useful if one has special devices in the field only used to route/forward packets in the network, acting similar to routers in the regular Internet, but not being able to e.g. respond to application requests. Nodes which do not use the application layer will reside in each nodes' routing tables, but should not show up as available nodes on the upper layers.

Proxy Certificate 0

A Service Proxy's Proxy Certificate 0 (PC0) would typically be filled with the values in the 3.2. For illustration, this PC has been issued by the SP's regular LLPKC issued by NTNU. The zero in the PC0 abbreviation is used to indicate that the

X.509 Field	Value
Subject Name	C=NO, L=Trondheim, O=NTNU, OU=ITEM, CN=Espen, CN=<SHA-1(Public Key)>
Issuer Name	C=NO, L=Trondheim, O=NTNU, OU=ITEM, CN=Espen
Serial Number	<SHA-1(Public Key)>
Key Usage	Digital signature, Key encipherment
pCPathLenConstraint	1
policyLanguage	id-ppl-anyLanguage
policy	Role:sp,Routing:full,Application:full

Table 3.2: Values in Proxy Certificate 0 (PC0)

certificate is at a depth of zero and has a pCPathLenConstraint of one, meaning that the SP is allowed to issue new PCs, but that the children of the PC0 cannot be used to issue new PCs again.

Proxy Certificate 1

If the PC0 above was used to issue a regular Proxy Certificate 1 (PC1) to an authenticated node, the values might look like the ones in Table 3.3. The interesting

X.509 Field	Value
Subject Name	C=NO, L=Trondheim, O=NTNU, OU=ITEM, CN=Espen, CN=<SHA-1(PubKey(SP))>, CN=<SHA-1(Public Key)>
Issuer Name	C=NO, L=Trondheim, O=NTNU, OU=ITEM, CN=Espen, CN=<SHA-1(PubKey(SP))>
Serial Number	<SHA-1(Public Key)>
Key Usage	Digital signature, Key encipherment
pCPathLenConstraint	0
policyLanguage	id-ppl-anyLanguage
policy	Role:authenticated,Routing:full,Application:full

Table 3.3: Values in Proxy Certificate 1 (PC1) if issued by the PC0 above.

part here is too see that the subject name contains the full subject name of the issuer in addition to the hash value of the public key used in the PC1. Note also that the pCPathLenConstraint now denies the owner to issue other PCs using this certificate.

Because the PC1 containing certain rights are signed by the SP, the holder of the PC1 cannot change the rights without voiding the digital signature. This property then assures requirements R6 and R7 from Table 3.1.

3.7.2 Service Proxy

The term Service Proxy (SP) was coined by Dr. Lawrie Brown of UNSW@ADFA [Bro10]. SPs are used in place of regular CAs for PCs. The SP determines whether a node should be issued a PC and if so which policies to attach. The SP drastically distinguishes itself from regular CAs because it breaks the hierarchical model usually associated with CAs when they are part of a PKI.

A SP can also be part of a PKI (issued a regular public key certificate), but because of our scenario, it can be difficult or impossible to verify its regular certificate. Therefore, this verification is handled out-of-band instead.

In this design, the SP is a master node which decides which nodes can get access to the network and if so what rights they have, fulfilling requirement R3 from Table 3.1. The SP assigns its PC0 with all the rights which might be useful for the current scenario, so that it can delegate those rights to other nodes when issuing them PC1s. In a larger network, there would typically be more than one SP

3.7.3 Authentication List

The Authentication List (AL) is a local list that each trusted node including the SP in the network maintain to keep track of which nodes they know and the necessary information about them, as shown in Table 3.4. An AL stores these values for each node they have met and shared their PCs with: Both the "ID" and "Subject Name"

Authentication List
ID
Address
Role
Subject Name
Public Key

Table 3.4: Authentication List (AL) content

fields stored for each node are unique, so one could ask why one needs the ID field when the unique subject name is used in accordance with RFC3820 ([TET$^+$04]). The answer is actually quite simple, it is easier and safer to look for the ID value when searching in the AL because it is a number while the subject name value is a text string.

For all nodes but the SP in the network, the first entry in the AL is the SP. This is only natural because the SP is the first node they start to trust as it is the one node that actually issued their PC in the first place. To illustrate this, if you go back to the two nodes' PCs in Tables 3.2 and 3.3 and you have a third node with these two nodes in its AL, the AL might look like Table 3.5. Note that the arbitrarily chosen

Fields	**Values**
ID	45251
Address	192.168.57.1
Role	SP
Subject Name	C=NO, L=Trondheim, O=NTNU, OU=ITEM, CN=Espen, CN=<SHA-1(2AF2E5C75C...)>
Public Key	2AF2E5C75C...
ID	1341
Address	192.168.57.45
Role	Authenticated
Subject Name	C=NO, L=Trondheim, O=NTNU, OU=ITEM, CN=Espen, CN=<SHA-1(2AF2E5C75C...)>, CN=<SHA-1(DA912BC9F5...)>
Public Key	DA912BC9F5...

Table 3.5: An Authentication List (AL) in a network with three trusted nodes.

values for e.g. the Public Key here does not show a complete size public key. Also, the public keys stored in the AL would be stored as their real binary values, and not Base64-encoded as the table portrays. In the subject name there is a common name part with "SHA-1()" - meaning that the content here would actually be the sha-1 digest of the content inside the parentheses.

3.7.4 Neighbor List

The Neighbor List (NL) is a list each node in the network maintains to keep track of its current neighbors and the necessary information about them. Table 3.6 shows the content fields for each entry in the NL. The first important thing to recognize is that the node entries in the NL are not necessarily the same as the node entries in the AL. When a node meets new neighbors and looses old direct neighbors (which are still in the network nevertheless) new entries are added while old entries are removed. This means also that the index in the two lists are not the same - and here's where the use of the ID field comes in. Whenever both lists has to be looked up, the ID field is used in order to retrieve the same node from both lists.

Besides the ID and the address the keystream field should be expected to reside

Neighbor List
ID
Address
Sliding window
Last sequence number
Keystream
Number of wrong OTPs
Time keystream received

Table 3.6: Neighbor List (NL) content

in the NL by the observant reader. If not, this is where the keystreams of your neighbors are stored, and updated each time you receive a new keystream-material message from said neighbor. There are four other fields too however, which might not be expected by the reader. These fields play an important part in maintaining this list and most importantly - replay attack protection.

First, the “Time keystream received” field stores the time (in seconds) when the last keystream from this neighbor was received. The Authentication Module regularly checks to see if it has not received keystreams from nodes in the NL for some time, and purges all entries older than a defined time frame. This is mainly to keep the NL up to date and short, as the keystreams take up memory storage.

The remaining three fields are quite more interesting, as they are used for replay attack protection.

Sliding Window

The sliding window is a bit array of 64 bits size [PD07]. This window is used in order to decide whether a routing announcement should be allowed through to the routing protocol, or if the AM should block and drop the announcement. Each bit in the window can either be 0 or 1, and it is used to declare whether an One-Time Password (OTP) already has been received (bit set to 1) or not (bit set to 0).

The “last sequence number” field is used to indicate the last, or highest, keystream sequence number of the routing announcements received from this neighbor. If the last sequence number is X, then the sliding window corresponds to the the sequence numbers between X-63 and X. I.e. the sliding window shows whether an OTP from this range has been received or not.

For a node to accept a routing announcement from a direct neighbor (which is in its NL), the following MUST be true:

- the OTP sequence number must either be inside the sliding window, or higher than the current last sequence number, and

- the same sequence number must not have been received before, i.e. the bit at its sliding window position must be 0, and
- the OTP must match the keystream at this sequence number offset.

Note that when a new keystream is calculated for a neighbor, the old sliding window and last sequence number are reset so the same sequence numbers can be used again.

The last field, "Number of wrong OTPs", is increased whenever a received routing announcement fails the checks above, i.e. if its an too old sequence number, it has been received before, or if the OTP does not match the expected OTP from the keystream. If this value is too high, the node will delete the neighbor entry from its NL and request a new keystream-material message. This value is reset to zero when a routing announcement from the same neighbor pass all the checks above, assuming the problem has been fixed.

3.8 Authentication Module Messages

This section describes the different messages sent within, or modified by, the AM extension. This includes how they are created, what payload they contain, if and how the information is secured from malicious actors, and to whom and how often they are sent during normal operation.

Each AM message has a header field which declares what kind of message follows in the payload and the unique ID of the sender. Some messages contain the exact same payload, but might have different message identifiers because the message can be sent by different purposes.

3.8.1 Node Discovery

Node discovery is not actually a part of the AM itself, but it is the main event that triggers the AM and is described here to make the context more clear to the reader.

Whenever a trusted node receives a routing announcement containing an One-Time Password (OTP) from a new neighbor, i.e. a node which is not in the Neighbor List (NL), the trusted node needs to check the Authentication List (AL) to see if the node is a trusted node. If the node can be found in the AL the trusted node will request and send its keystream-material message.

If the neighbor is not found in the AL however, the node will send its PC and request the neighbor's PC. In addition, both nodes will after verifying the other nodes' PC send its keystream-material to the other.

If the routing announcement does not contain an OTP at all, meaning the neighbor is either a new or untrusted, regular authenticated nodes will ignore and drop it as they do not take part in the authentication of new nodes. If the trusted node is also the Service Proxy (SP) however, it will invite the unknown neighbor to an authentication handshake before dropping the packet.

3.8.2 Authentication Handshake

The authentication handshake describes which messages are sent while authenticating a new node to the network. If the handshake is successful the two nodes participating in the handshake will share their keystream-materials with one another.

Handshake Invite

The handshake invite message is a rather simple message. Its header declares that it is an “invite message”, and the payload only contains the SP’s PC. The PC is sent in the beginning in order for the new node to be able to verify the certificate before engaging in the handshake.

Proxy Certificate Request

This message contains the field declaring it is a “certificate request message” and its payload contains a X.509_REQ data structure [VMC02]. This data structure is essentially a certificate without an issuer’s signature, containing all the values it wishes the SP to grant it.

The request data structure also contains the extensions being used, and specifically the sender will request certain rights through the policy field of the proxy certificate. E.g. it can request full routing and full application rights as mentioned in Section 3.7.1.

Proxy Certificate Issue

Completing the handshake is a message containing the signed certificate. If the SP was able to verify the new node’s public key, the SP would sign the request in addition to adding some values, possibly changing some of the requested values, and sending it all in one message.

After this message, the SP waits for the first keystream-material message from the newly authenticated node as an acknowledgment from the recipient, after which the

SP sends its keystream-material to the new node as well.

3.8.3 Keystream-Material Message

This message contains two headers - a regular AM header declaring the type of message and ID of the sender, and a special header declaring the different lengths of the contents in the payload.

The payload contains everything needed to create the sender's keystream, i.e. a nonce, an ephemeral key encrypted with the recipients public key, an IV, and a digital signature. The signature is used to prove the authenticity of the message, i.e. to prove the integrity of the message content and to authenticate the sender - or prove the sender is who it claims to be. The digital signature is created by creating a message digest of the nonce, IV, and unencrypted (!) ephemeral key and encrypting the message digest with the senders private key.

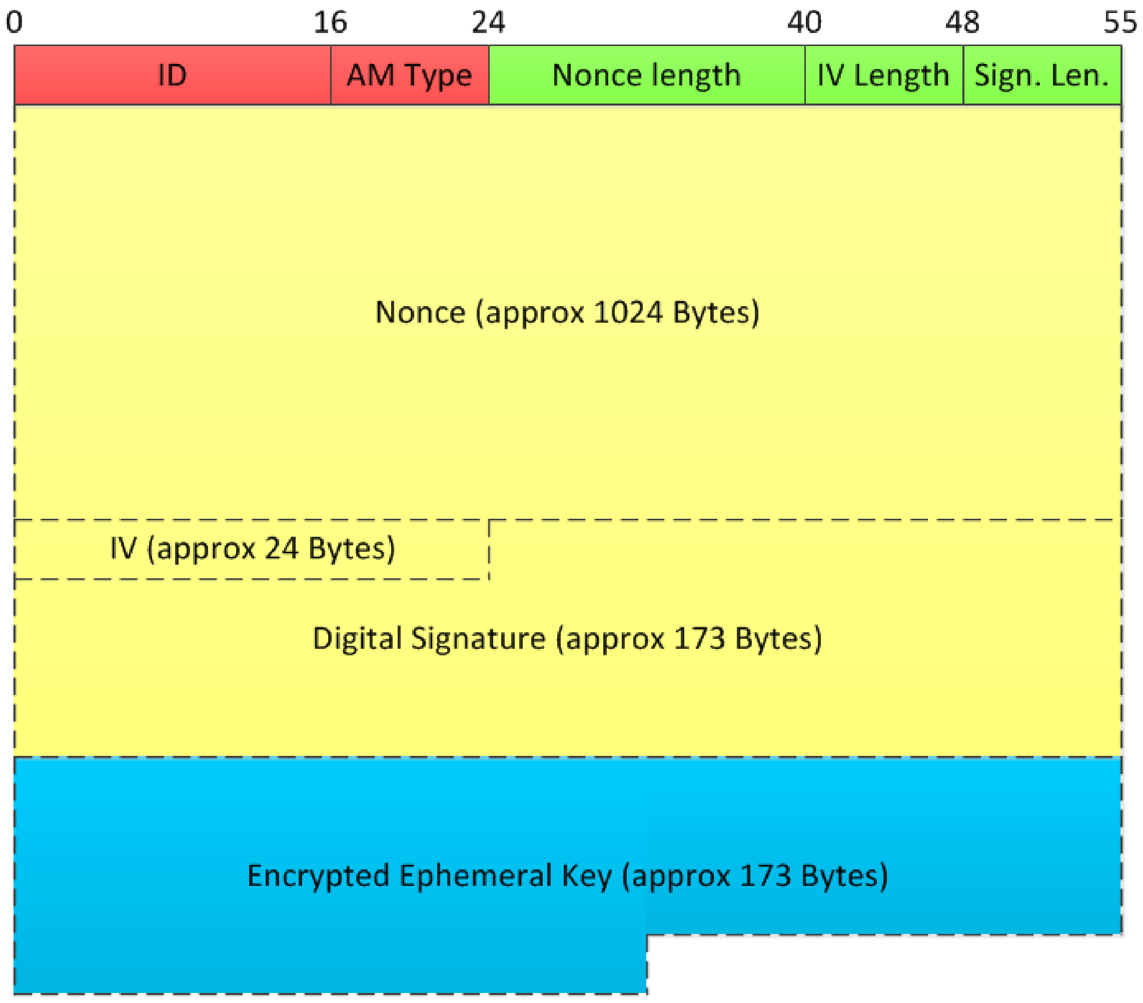

0
16
24
40
48
55
ID
AM Type
Nonce length
IV Length
Sign. Len.
Nonce (approx 1024 Bytes)
IV (approx 24 Bytes)
Digital Signature (approx 173 Bytes)
Encrypted Ephemeral Key (approx 173 Bytes)

Figure 3.7: Packet format for keystream-material messages.

As can be seen in Figure 3.7, the encrypted ephemeral key is appended at the end. This, and the decision to make the digital signature from the unencrypted key, is done in order to make the packet "re-usable" - remember this packet has to be sent by the creator to each of its neighbors, and with this design only the ephemeral key needs to be encrypted and appended to the "standard" packet for each recipient. This not only makes the design easier, it saves overhead by not having to create a digital signature for each recipient of the packet. In this figure it is assumed that the nonce is 767 bits long, the IV 128 bits, and the RSA keys used to encrypt create the signature and to encrypt the ephemeral key are 1024 bits. The additional bytes come from being Base64 encoded in the message.

This message is sent to all neighbors periodically (changing the keystream), or when the keystream has been exhausted - whichever comes first.

Keystream-Material Request

There is a special case the keystream-material sharing. If a node continuously receives routing announcements with bad OTPs, the node will assume the neighbor has created a new keystream but its keystream-material message has not been successfully received by this node. Because this often happens if the packet delivery ratio is very low, the node also assumes that the other neighbor does not have its keystream-material.

The node therefore generates a regular keystream-material message, only different by the first header type declaring this to be a "keystream request", and sends it to its neighbor as if it was a new neighbor. Note that no new keystream-material is created here, it is only a "re-sending" of the old packet appended with the current ephemeral key (encrypted).

3.8.4 Modified Routing Announcements

The routing announcements are sent by the original routing protocol and not by the AM, but a small but significant alteration to the routing protocol has to be made in order to append the OTPs to the routing announcements.

The routing announcements are appended with 16 bits extracts from the keystream, called one-time-passwords. Additionally, each routing announcement is also appended with a sequence number for which the offset of the keystream the OTP matches. In the next chapter Figure 4.2 shows how the routing announcements for BATMAN, which are called Originator Messages (OGMs), are modified with these OTPs.

Figure 3.8 shows how the first five routing announcements, after a new keystream is shared with neighbors, are appended with the five first OTPs from the new

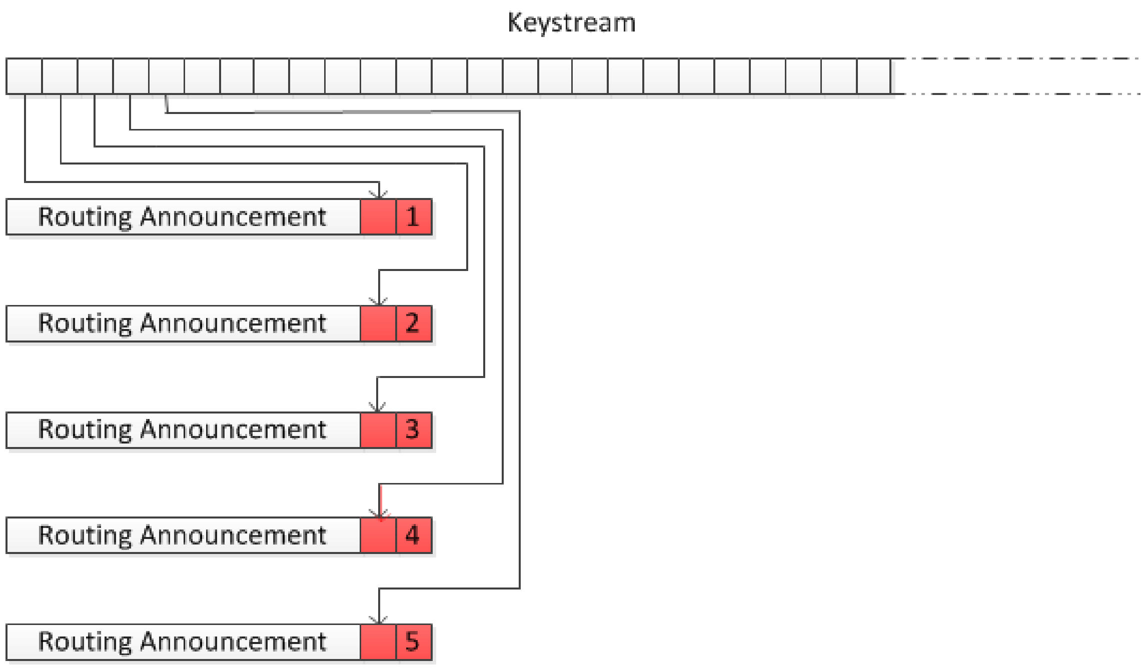


Figure 3.8: One-time passwords and their sequence numbers appended to routing announcements.

keystream. Additionally it shows the authentication sequence number appended after the OTPs again, required by the recipient to know which OTP to match in the sender's keystream.

For each routing announcement a neighbor receives, it checks the OTP at the given offset of the senders keystream and if it matches - it assumes the message is from its trusted neighbor. If the OTP does not match the packet will be dropped as discussed earlier.

Chapter 4

Implementation

This chapter goes into depth on how the design from the previous chapter is implemented into the BATMAN protocol. Some code snippets will be shown in the following sections when applicable, and the full source code can be found in Appendix A.

4.1 OpenSSL Library

All of the cryptographic functions such as encryptions, signatures, and X.509 public key certificate creation and verification are created using functions from the OpenSSL library [VMC02]. This library was not in the original implementation of BATMAN, and has to be installed on the computers running this modified version and added to the Makefile for the BATMAN implementation. How to install the correct OpenSSL library in a Debian environment is explained in Appendix B, and the modified Makefile is part of the full source code linked to in Appendix A.

4.2 Authentication Module

Almost all functionality added to BATMAN is within the borders of a separate class called Authentication Module (AM). The first thing to notice about the AM is that it runs in its own thread and sockets, so all authentication mechanisms run concurrent to regular BATMAN routing operations. This separation was necessary in order to have BATMAN behave normally during e.g. the authentication of a node, so a large network should not suffer if an important node (centrally located) is 'hung up' in e.g. authenticating another node.

4.2.1 AM Thread

In the setup phase in the original batman class, an AM initiation function called `am_thread_init` from the AM class is called. This function takes the network interface name and its corresponding IP address and broadcast address as input. These values are then stored locally in in the AM class for socket setup. For socket setup see Section 4.2.2.

The function then goes on to create a new thread for the AM module which is the main thread taking care of most of the additions in this implementation. The thread first sets up two sockets for sending and receiving AM messages such as handshakes and keystream-materials.

Next it generates a highly random (high entropy) master key using the OpenSSL function `RAND_bytes`, which has been properly seeded - see Section 4.8.1. This and an IV generated with OpenSSL's `RAND_pseudo_bytes` is then used to generate a master key encryption context for AES encryption, before deleting the master key. With the encryption context the necessary internal memory used by OpenSSL for encrypting with this key is stored, and therefore the key and IV is of no more use by themselves - making it a sound security choice to delete the key (and IV) entirely.

The next important action is to generate either a Proxy Certificate 0 (PC0) or a Proxy Certificate (PC) request depending on whether you are a Service Proxy (SP) or just a regular node trying to authenticate with the network.

After these steps the "initiation phase" of the AM thread is complete, and the rest of the code runs in a loop until the BATMAN daemon is terminated.

4.2.2 AM Sockets

Two sockets are used for the AM module, one for sending and one for receiving AM messages. Both sockets are bound to the interface device given by the AM initiation function. The receive socket is then bound to a designated port 64305, regular BATMAN runs on port 4305, and the send socket is explicitly allowed to send to broadcast addresses. Sockets needs to be explicitly set to be allowed to send to broadcast addresses in UNIX systems, as a protection mechanism. A code snippet of the sockets set up is shown in the appendix under Section A.2.1.

There are several practical reasons to choose UDP sockets over TCP sockets for this implementation. First and foremost, this system sends authentication handshake messages and keystream-material messages to nodes which has no route in the routing tables. If a connection-oriented protocol would try this the messages would be blocked on the kernel level and not sent. With an connectionless protocol like UDP no mechanisms will block this message being sent, it will just send the message not bothering whether the message is ever received by the recipient.

Second, TCP was created with wired networks in mind, observing much less packet loss. In Mobile Ad Hoc Networks (MANETs) the packet losses are much higher than in wired and fixed infrastructure networks, and as nodes move around direct paths between nodes change much more frequently. TCP is not suitable for such environments because it will lead to a huge amount of re-sending of packets to "non-existent" neighbors and much memory wasted in connection states being kept for dead links.

4.2.3 Main Operation of the AM Thread

Most of the interesting operation in the AM class happens within a single loop running throughout the lifetime of the BATMAN daemon. The program flow throughout this loop is shown in Figure 4.1.

For clarity the figure leaves out certain details, but the most important features are depicted. The handle incoming messages are shown as a sub-process in the figure. This is not true, but depicting all possible messages would not fit the figure, and therefore left out altogether. However, each message are described in the following sections.

The "New Neighbor" is one of the elements in the AM class that must be triggered by the batman class. If the batman thread tells the AM thread a new neighbor is discovered and the AM thread is in a ready state to handle new neighbors the appropriate action is taken, whether the neighbor has been authenticated with the network or not. Not shown in the figure is how a regular authenticated node will act if a new neighbor which is not authenticated with the network is handled, but this is taken care of in the batman class and not here.

The two last parts should be self-explanatory, they simply check the current time and then compares this against time values set on the nodes own current keystream to check if its old, or if any keystreams from other neighbors in the Neighbor List (NL) are old and if so takes the appropriate action. Also checked is if a nodes own keystream is getting exhausted, in which also the appropriate action, namely creating and sharing a new one with each neighbor in the NL.

4.3 Proxy Certificates

The PCs in this implementation are containers for short lived 1024 bits RSA public keys used in a single session only. Most of the design choices were taken into the implementation, but some however, were left due to time constraint.

One such was that the original proxy certificate X.509v3 extension, introduced in RFC 3820 called "proxyCertInfoExtension" was not used to carry the policies as

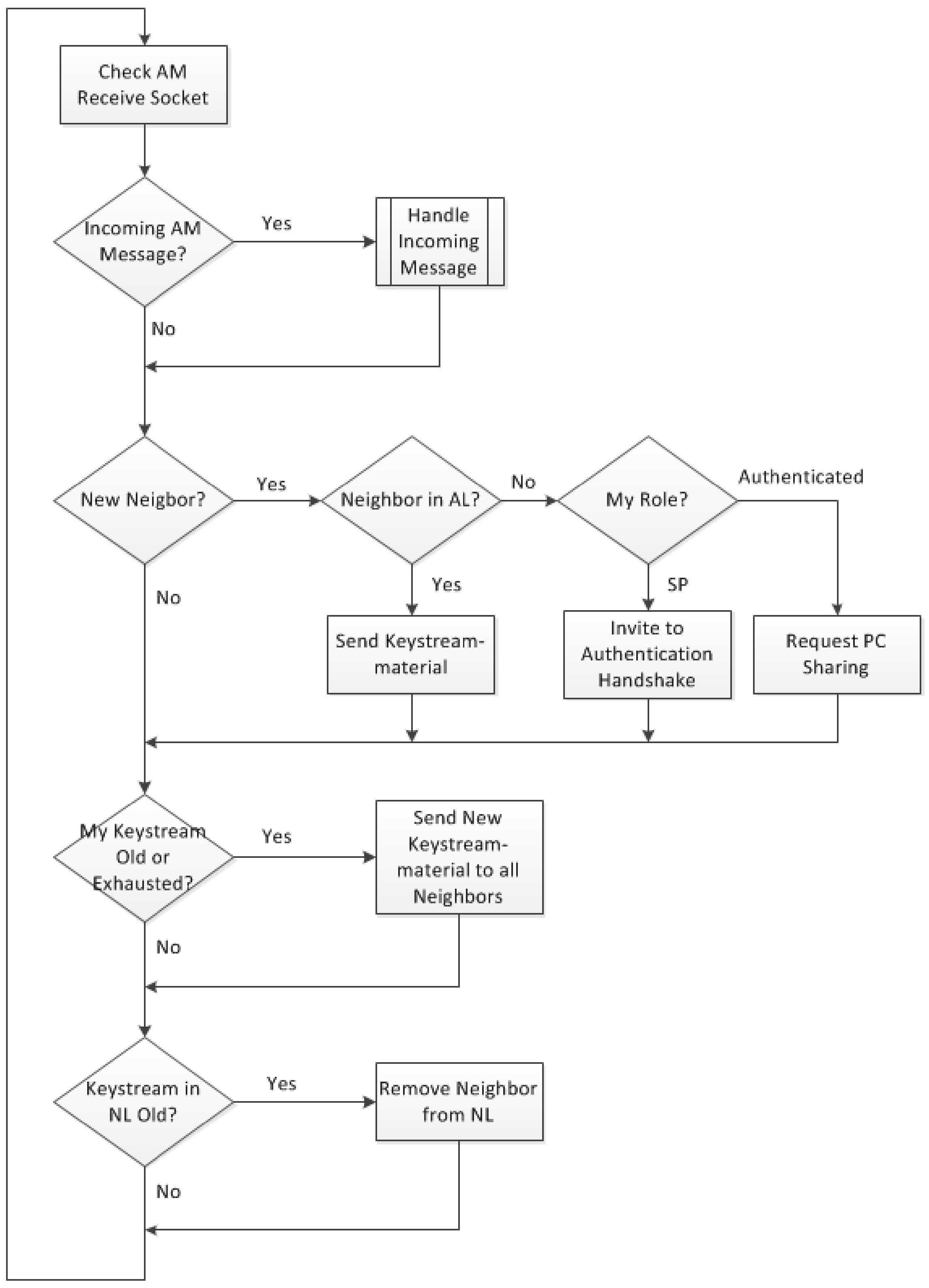


Figure 4.1: Main program flow in AM class.

intended. Most of the OpenSSL documentation is not released to the general public for free, and this X.509v3 extension was no exception. The only examples found, amongst the original proxy certificate implementations from the Globus Project[1], but using the exact same setup did not work in my implementation. After some investigation it seems no open source code projects using proxy certificates have been published for years, giving me the idea that maybe the OpenSSL specifications have changed during the last version updates and that proxy certificates have to be implemented differently. The author have asked for an answer on this subject on both emails to the developers of OpenSSL, the OpenSSL's mailing lists, and on an OpenSSL "IRC" channel[2].

As a replacement for this extension, a commonly used free-text extension called `netscape_comment` has been used and the policy has been written in cleartext inside this comment. Because the design proposed used the `id-ppl-anyLanguage` and allows the application to decide the language the policy is written in, this should make no practical difference, other than not being a strictly RFC 3820 proxyCertInfoExtension.

4.3.1 Generating PC Requests

As mentioned earlier the PC requests are generated prior to the main loop of the AM thread. This means the request is generated and ready prior to discovering new neighbors, so the authentication handshake can be performed as quick as possible.

The request starts with creating and assigning a RSA key pair the size of 1024 bits. If the reader take a look into the implementation he or she will see that the setup for an Elliptic-Curve Cryptography (ECC) key pair is also there. Using ECC was only scrapped at the very end because of problems regarding the keystream-material exchange later on, where the author had problems with the ECIES algorithm.

The subject name is created a pseudo random function, contrary to using the SHA-1 digest of the public key as proposed in the design. This was implemented this way at the beginning quickly, and was supposed to be exchanged with the message digest later on but because of time constraint there were no time to test whether this subject name was set correctly. It should however be an easy and relatively quick fix, but needs to be tested properly first - and probably be encoded in Base64.

After this the proxy certificate extension is added using the following function `openssl_cert_add_ext_req` as seen in Section A.2.2.

All this data is stored as a X.509_REQ object and written to disk and sent during the handshake in a PEM encoded format [Lin93].

[1]Globus Project: http://www.globus.org/toolkit/downloads/
[2]OpenSSL IRC Channel: #openssl on irc://irc.freenode.net

4.3.2 Generating PCs

Generating PCs are much the same as creating PC requests, with the addition of signing the certificate, changing the subject name, setting the issuer name, and setting the valid lifetime of the certificate. The issuer name is simply set to the subject name of the SP and the subject name is set as shown in Section A.2.3. The valid lifetime is set from the current time, and until 8 hours from creation.

4.3.3 Verifying PCs

In this implementation PCs and their public keys are not verified during the initial authentication phase. This implementation use the physical locality as the "out-of-band verification", which might be good enough if the SP only use an ethernet cable which has to be directly connected to the new nodes trying to authenticate themselves. However, in a real-world implementation this method has to be revised!

However, when the initial authentication with the SP is complete, nodes do verify each others' PCs by checking the signature against the SPs public key using `X509_verify` taking the received certificate and the SPs public key as input.

4.4 Authentication List

When a node has verified another node, it stores some information about that node within a struct called `trusted_node`. Then the struct is placed in a list called Authentication List (AL) which stores one such struct for each known and trusted node. Table 4.1 shows how this data structure looks like: At the beginning of the

Data type	**Variable Name**
uint16_t	id
uint32_t	addr
uint8_t	role
unsigned char *	name
EVP_PKEY *	pub_key

Table 4.1: Trusted node struct of the AL.

project, the AL was intended to be sent on the network and was therefore made as an array, instead of a linked list. With additional time this list should however be converted to a linked list for better performance and less memory waste. The complete code snippet showing how a newly verified node is added to the AL is shown in Section A.2.4.

4.5 Neighbor List

Similarly when a node receives a new keystream-material message from a neighbor the neighbor is added to the NL. This list is also an array, and should also be changed to a linked list, which contains a data structure called `trusted_neigh`. Below Table 4.2 shows this new data structure: Note here that while the design and

Data type	**Variable Name**
uint16_t	id
uint32_t	addr
uint64_t	window
uint16_t	last_seq_num
unsigned char *	mac
time_t	last_rcvd_time
uint8_t	num_keystream_fails

Table 4.2: Trusted neighbor struct of the NL.

implementation changed during this thesis things have changed, and what used to be a Message Authentication Code (MAC) in the NL is now a keystream, but the names have not been changed in the current version of the implementation.

The code snippet in Section A.2.5 shows what happens whenever a new neighbor is added, or a new keystream-material is received from a neighbor. The code snippet in Section A.2.6 shows what happens whenever a node is removed from the NL due to inactivity, i.e. not received its keystream-material for 130 seconds.

4.6 Keystream Generation

Before a keystream can be generated, its keystream-material must be generated (and sent). The IV and nonce are both generated using OpenSSL's `RAND_pseudo_bytes` generating 16 and 767 Bytes of pseudo random data, respectively. The generation of the ephemeral key is created using AES-CBC as shown in the code snippet in Section A.2.7.

When the keystream-material has been created, they can be used to create the full keystream on both ends. First the original nonce is encrypted using the ephemeral key and IV as inputs to OpenSSL's AES-CBC encryption. Next one tenth (1/10) of the nonce will be XOR'ed with 1 and encrypted again with the AES-CBC encryption, using the same key but using a part of the previous ciphertext as IV. This is done 9 times, so the second round the second part (between 2/10 and 3/10) of the nonce will be XOR'ed with 2 and encrypted with the same key and IV from previous ciphertext and so on until ten different (1 original and 9 modified) nonces has been encrypted.

All the ciphertexts is then added to one large buffer, which is this node's keystream. All of the operations above is shown in the code snippet in Section A.2.8.

4.7 Using One-Time Passwords

One-Time Passwords (OTPs) are handled within the context of the original BATMAN protocol because adding and verifying them is performed when the original protocol receives or sends them. How to add the OTPs are described in Section 4.8.3 while how to verify them are described in Section 4.8.2.

4.8 Changes to the BATMAN Protocol

Some changes are BATMAN specific or needs to be called from within the original BATMAN protocol and these changes are described here.

Figure 4.2 shows how the modified batman packets, or Originator Messages (OGMs), look like. The white area is not changed, except an unused version number is taken and used in the 'Version' field at the outset of the packet. The red area is added by this implementation, containing both an OTP and its sequence number in the keystream.

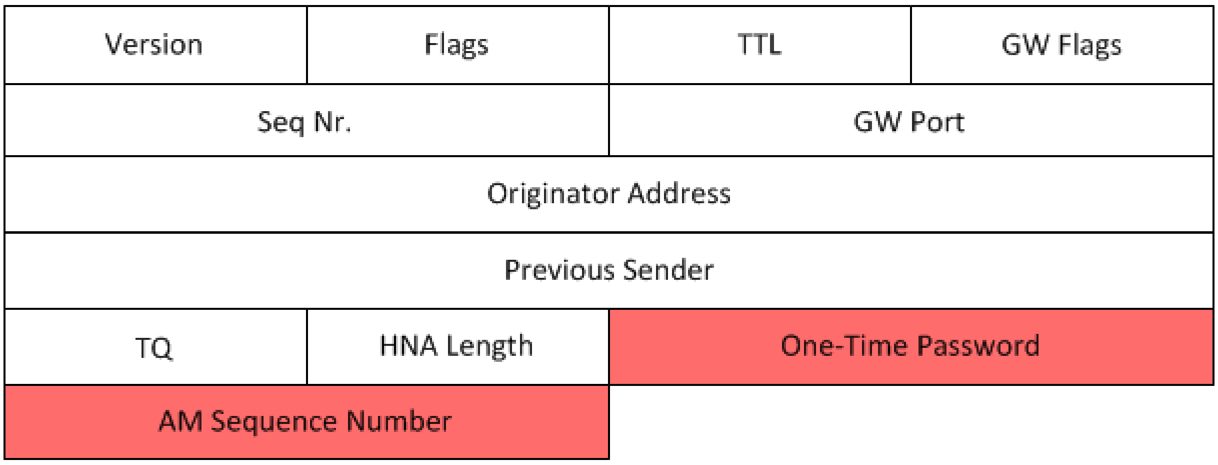


Figure 4.2: Modified routing announcement (OGM) for extended BATMAN.

4.8.1 POSIX.C

It is within the posix.c class that the daemon's main function that initialized the whole program lies. In here a small but important modification was made. Before allowing the daemon to start, OpenSSL's pseudo random number generator (PRNG) has to be seeded with much high entropy random data. This is done by taking 1024 bytes from the UNIX environment's */dev/urandom*. If this seeding should fail, the daemon is not started and the program killed.

4.8.2 BATMAN.C

It is in this class where the main thread in the original BATMAN runs, where it receives OGMs, decides how to handle them, and whether to schedule to send own or re-broadcast other nodes' OGMs. Before entering the main loop, this class has been modified to initiate the AM class by calling `am_thread_init`.

Inside the main loop which is loops for each received OGM there has been a modification to verify the OTP appended to the OGM. If the verification check is not passed, the OGM is dropped, and possibly the AM class is told there is a new node. The interesting code snippet from this class can be found in Section A.2.9 in it entirety.

Figure 4.3 shows the flow chart through the main loop in BATMAN. Here you can see where the AM modifications are put, and hopefully understand why they are put there. The first checks before the modification are almost like sanity checks, which must be passed before testing the OTPs should even be considered. After the AM extension there are many checks which are more specific to the routing algorithm, but they do not affect whether the OTPs should be checked, and if the OGM is received from a new neighbor the AM extension has to notify the AM class of a new neighbor before allowing the the subsequent checks to be made, or the OGM will be handled when it quite possibly should not.

4.8.3 SCHEDULE.C

In the schedule class there are two functions which needs to be modified. The two functions `schedule_forward_packet` and `schedule_own_packet` generates and puts OGMs in a queue waiting to be send on behalf of other nodes or itself, respectively. The modifications are the same using only different available parameters, and is shown in Section A.2.10.

Again there is some variable name discrepancy because the design and implementation has evolved during the work of this thesis. The `auth` variable name used actually means OTP and the `auth_value` means the whole keystream. The line below means that a value of 2 bytes size is to be copied from the keystream to the OTP value in the bat_packet, or OGM.
`memcpy(bat_packet->auth, auth_value+2*auth_seq_num, 2);`.

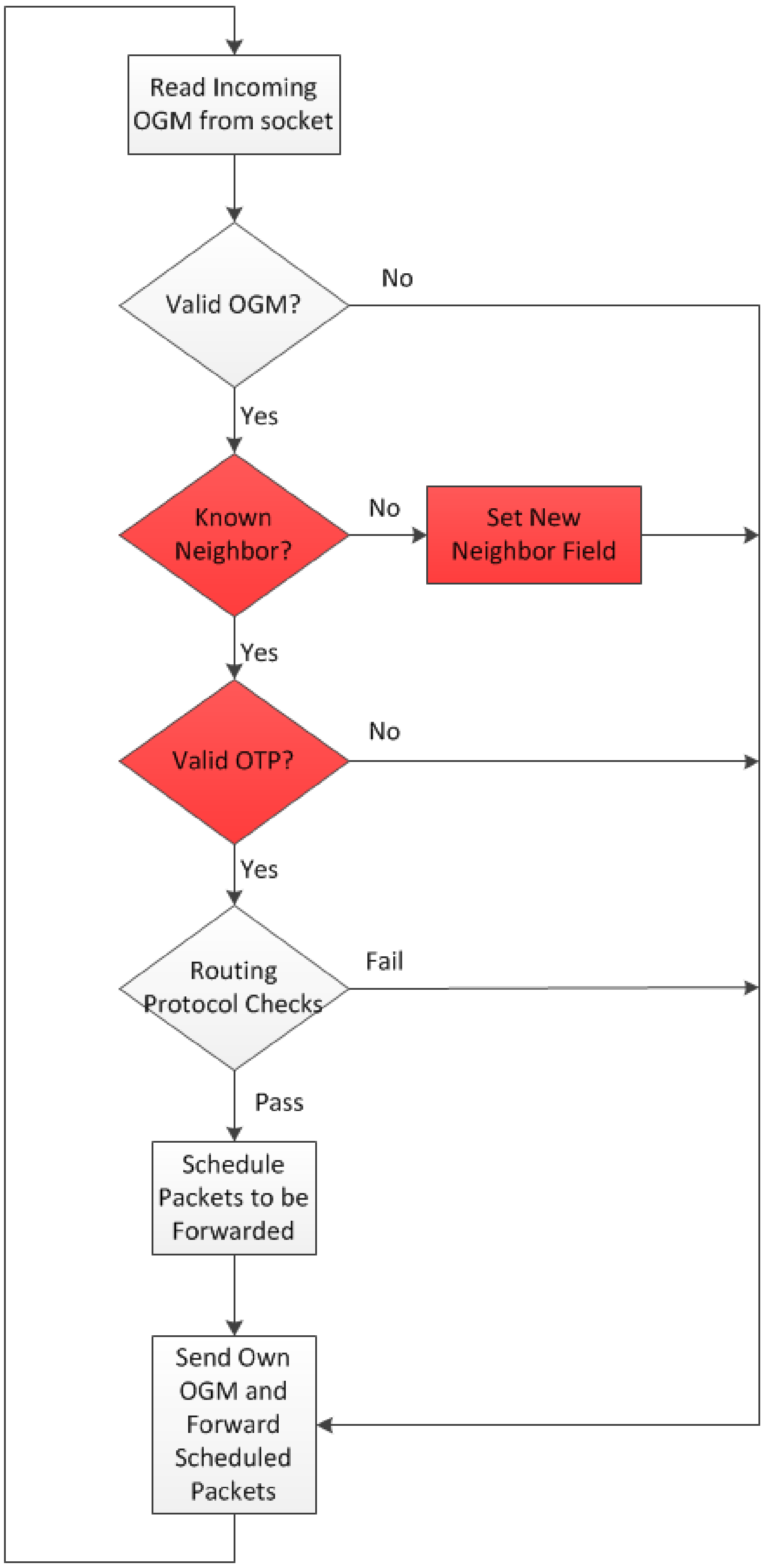

Read Incoming OGM from socket
Valid OGM?
No
Yes
Known Neighbor?
No
Set New Neighbor Field
Yes
Valid OTP?
No
Yes
Routing Protocol Checks
Fail
Pass
Schedule Packets to be Forwarded
Send Own OGM and Forward Scheduled Packets

Figure 4.3: IF-Statements in the Batman Class

Chapter 5

Testing & Results

In this chapter two tests are described and their results presented and discussed. The two tests measure and compare the time performance in two common stages for both the original implementation of BATMAN, and the extended version proposed and implemented in this thesis.

5.1 Test I - Initialization Phase

The “initialization phase” is the setup phase between two or more nodes trying to create a network. With the original implementation of BATMAN this phase only consist of two stages; namely discovering a neighbor node, and deciding to add the node as a direct link, or “last-hop” per BATMAN terminology, in its routing table.

With the proposed design and implementation from this thesis, two more stages are added. After the discovery, the authentication handshake stage and the keystream sharing stage are conducted before the last stage where BATMAN adds the node as a new direct link in its routing table.

The time measured here in this test is the time between the first discovery of a new neighbor, until that node is added to the routing table.

5.1.1 Hypothesis

With the modified version of BATMAN proposed in this thesis, one should observe a small extra delay in the setup of the network, compared to the original BATMAN protocol. This extra delay should however, not be significantly higher, i.e. it should be relatively constant and at no time should any linear increase in delay be observed.

5.1.2 Setup

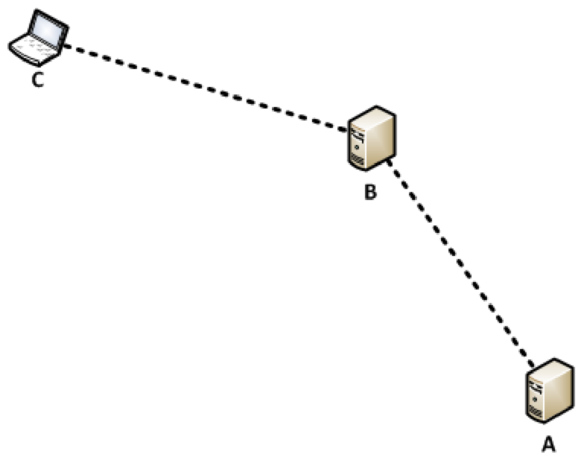


Figure 5.1: Physical network layout used in test 1. When using the modified version node B acts as the SP of the network

Figure 5.1 presents the setup of the test machines used to conduct this first test. Node A and B are stationary boxes while node C is a laptop. Their hardware specifications are described in Appendix B. The reasoning to use a different hardware for node C is the need to create distance in the network, and that outside the ethernet subnet for which the two other nodes were connected to, it would be easier to use a laptop during setup. In the next test, this laptop is yet again moved further away.

An important feature to notice about how these nodes were set up is that node A and C are outside each other transmitting range, meaning they need an intermediate node to route their packets to and from each other. Node B is conveniently placed with almost equal distance to each of the two other nodes.

The landscape the nodes are setup in is a typical office landscape, with varying obstructing materials such as concrete, wood, and glass. A more ideal setup would naturally be outdoors, as the network is intended for, but with the lack of mobile nodes and time this became out of the option.

5.1.3 Procedure

In order to get the same behavior each run for the modified version, each run had to be run discretely, i.e. after each run the daemon was shut down and restarted. This way each run will include all four stages explained above: discovery, authentication handshake, keystream material sharing, and routing table update. This was also done on the original implementation, even though there are no authentication steps in between, but in order to have the exact same procedure each time.

For each run, these steps were followed:

1. Start Node A and C
2. Wait and make sure both nodes are stable
3. Start Node B
4. When both node A and C are discovered and added to routing table kill all daemons
5. Record the log from node B

These steps were taken 10 times in order to have a reasonable data set and average. Then for each of the 10 logs, record the time between the first routing announcement received from a node, until both nodes have been added to the routing table.

5.2 Test II - Route Convergence

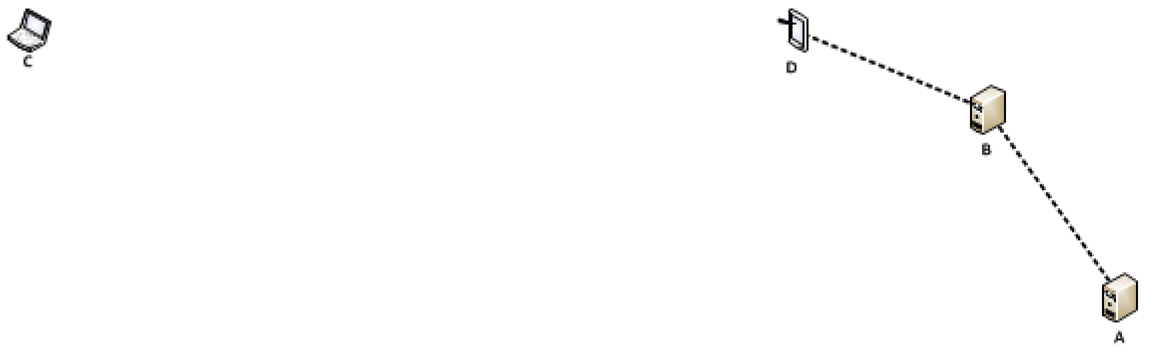
The next test is to check how quick route convergence the protocols deliver when a path between nodes in the network changes. As with the previous test, a node running the original protocol needs to discover a new direct neighbor and add it to its routing table. In addition, it will have to receive not only the neighbors own routing announcements, but also routing announcements it has forwarded on behalf of its own direct neighbors. When this happens, the node receiving these re-broadcasts determines new routes to possibly new nodes - adding them to its routing table.

Additionally, nodes using the modified BATMAN protocol needs to share their keystream material with their new direct neighbor before adding that neighbor in their routing table. When this has happened, any re-broadcasted routing announcements from that neighbor is accepted and paths to those nodes are updated, or added if new nodes. Note also that no authentication handshake or sharing of certificates are mentioned, as they are assumed to have been shared prior to this route change.

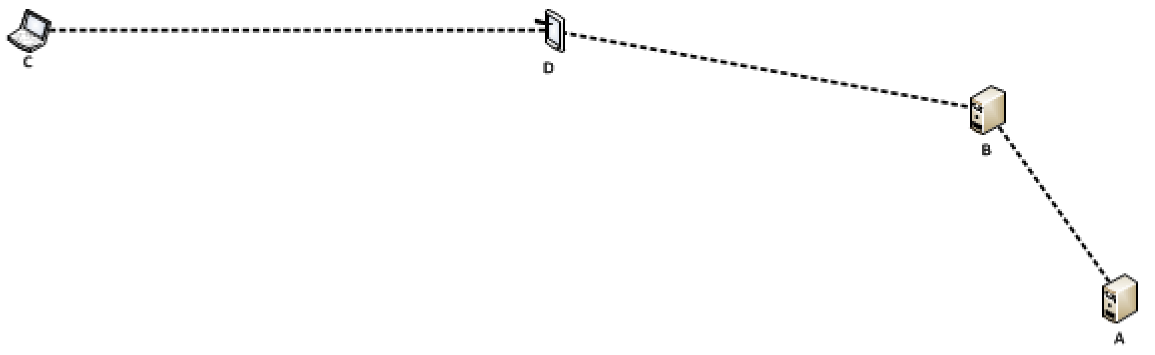
5.2.1 Hypothesis

As before, a small and relatively constant (mathematical term) extra delay is expected when running the modified version of the protocol compared to running the original. As the keystream material sharing only occurs between the direct neighbors, it should only happen once during one run and not for each of the new nodes added by the path through the direct neighbor - i.e. there should be no linear increase in convergence time even if multiple new nodes are added to the routing table with paths through a new neighbor.

5.2.2 Setup



(a) Initial Position - D out of range of C



(b) “Connected Position” - D within range of C

Figure 5.2: Physical network layout used in test 2. The Laptop (node C) is moved further out of range and is periodically rejoining the network when a Tablet Pc (node D) is moved within range. When using the modified version node B acts as the SP of the network

For this test a new mobile node was needed. As Figure 5.2 shows the Laptop, or node C, from the previous test is moved much further away, so far in fact the newly added Tablet Pc, node D, needs to place itself with approximately the same distance to node B as to node C in order for node C to take part in the network.

Node A and B is positioned at the same place as in the previous test, in the same office landscape. The line of sight is, however better between node B and D, and between D and C. Their line of sighs are only obstructed by a single wall with huge windows.

5.2.3 Procedure

With this test running each of the 10 iterations discretely would be too time consuming, because it would have meant that the Laptop (node C) would have to be physically moved inside the transmitting range of node B for each run. Instead the laptop was only started within the range of the other nodes, in order to be authorized and share certificates with the other nodes (only node B and D was necessary) and then moved to its position long outside the transmitting range of the rest of the network. After this “initial setup” plus some minutes to clear the neighbor lists, these steps were followed:

1. Walk node D in between node B and C
2. Wait until the whole network has stabilized
3. Walk node D back out of node C’s range
4. Wait until BATMAN has cleared node D from other nodes routing tables, and node D has cleared all other nodes from its routing table
5. If modified version is used, make also sure node D is removed from the Neighbor List (NL) (should be before routing tables are cleared)

These steps were repeated 10 times for both implementations. The records used from this test are from the logs of node C. The convergence times measured are the time between each time a new neighbor (node D) is discovered by BATMAN, and until a path to the furthermost node (A) is added to the routing table.

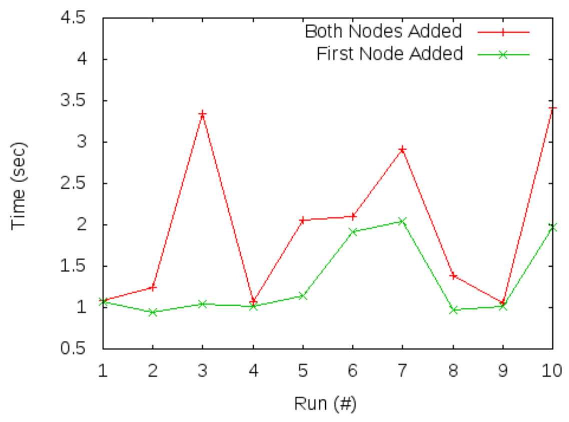
5.3 Results

In this section the results from the two tests above are presented and discussed in terms of how they perform compared to the hypotheses. In both tests there were only a dataset of 10 trials, which given the variance shown below does not provide a statistical significant result. However, the first test shows good indication that the modified protocol behaves as expected, while the results from the seconds test are more ambiguous. The results below are presented in graphs, while all of the numerical results and the test logs are found in Appendix C.

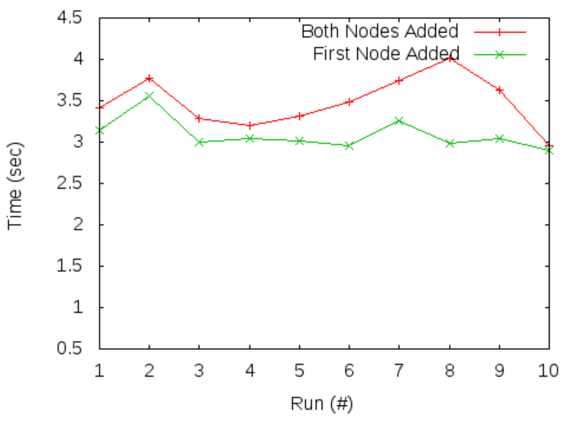
5.3.1 Initialization Phase

Figure 5.3 presents the first test’s results for both the original and modified version of BATMAN. The two graphs shows the time in seconds on the y-axis and the trial/run number on the x-axis. The two colored lines on the graphs shows the results from first neighbor discovery until the first neighbor is added to routing table (green line) and until both nodes are added to the routing table (red line).

The results from the original protocol, shown in Figure 5.3a, shows high variance in the time needed to add one and two nodes to the routing table. For 7 out of 10 “first



(a) Original B.A.T.M.A.N.



(b) Modified B.A.T.M.A.N.

Figure 5.3: In a network of three nodes, the time spent by the SP from its first neighbor discovery and until both neighbors are added to its routing table.

nodes" the time needed is relatively equal, being about one second. For both nodes to be added however, there are much more variance - varying from the best possible time, i.e. equal to adding one node, and up above 3 times longer than adding one node. As this is from the original implementation of BATMAN, this thesis will not try to explain why this behavior is observed, nor does the author know exactly why either.

Figure 5.3b shows the results from the modified version proposed in this thesis. These results indicates that the behavior of the modified version seems to correlate with the behavior expected from the hypothesis. A seemingly constant of about two seconds seems to be added to the process of adding both nodes to the routing table.

Another interesting observation is that the time variance seems to be much less from that of the original version. This might be because the authentication handshake and the keystream sharing happens in a separate thread from the regular BATMAN operations, meaning the BATMAN protocol continuously receives routing announcements to process while the Authentication Module (AM) handles its part. The idea being that while the AM thread runs the BATMAN thread "gets ready" to do its part of the job.

5.3.2 Route Convergence

The results of the second test is shown in Figure 5.4. In this figure, the axes are the same as in the figures above: y-axis shows the time in seconds, and the x-axis shows the trial run. The red line shows the performance of the original implementation, while the green line shows the modified.

As indicated earlier, this test's results are somewhat unclear. While the results

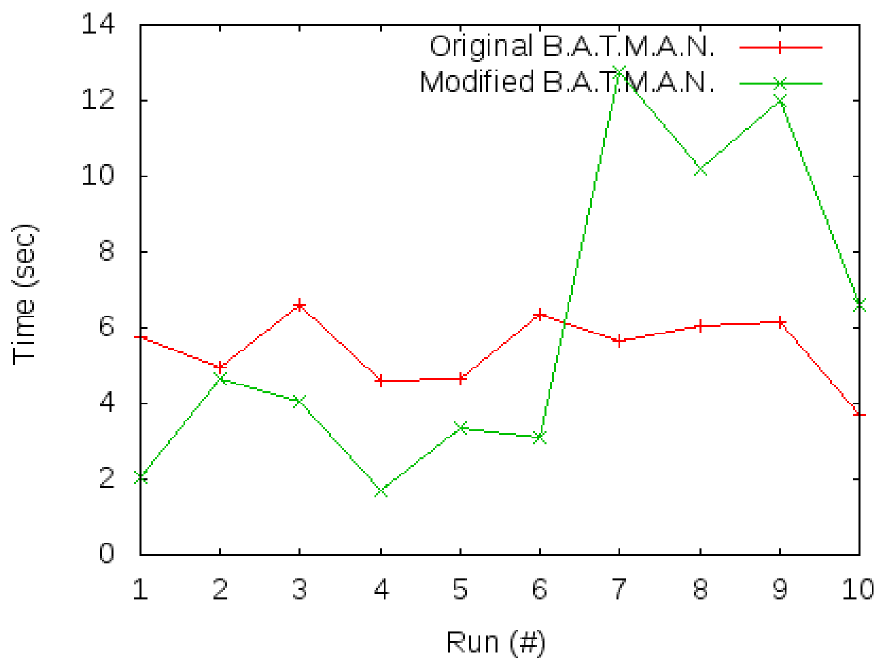


Figure 5.4: Routing path convergence time observed by a distant source node to another sink node in the network. The source node is only sporadically connected to the network through a mobile intermediate node.

using the original implementation seems relatively uniform, with only about 1 second variance, the results from the modified implementation is highly irregular.

Looking through the logs from this test one thing become apparent. With different hardware on the different nodes in the network, their wireless cards send at different strengths meaning while one node can receive packets from a “stronger node”, the packets sent might not be received by the other nodes.

The BATMAN protocol messages (routing announcements) are sent quite often, depending on the number of re-broadcasts being sent, meaning the time from when a node is within transmitting range and until its broadcasts are received by nodes within its transmitting range will be quite short. The AM messages however, was mostly tested in an ideal environment where most packets were received, so this was not properly accounted for. Therefore, if a routing announcement from a “stronger node” is received by a “weaker node”, the weaker node might send its keystream material without the other node receiving it.

Re-transmitting mechanisms based on guessing that the receiving node has not received the AM messages are in place, but as the mechanism wait until it believes the other node has not received, instead of knowing it instantly. This can of course be managed adding ACK’ing to each AM message, which was not added initially because of the wish to minimize overhead. This however, might have to be re-evaluated.

Another thing to notice is how multiple trial runs using the modified version actually performed better than the original version. This is impossible to explain talking about the design and implementations themselves, but is probably most accurately explained in the terms of external environment.

In this test, one major factor is the movement of the Tablet Pc, or node D. This movement is not perfect, and will vary in speed, timing, and accurate position for each run. As self-generated routing announcements are only generated once every second, a difference in almost two seconds can be seen based on the difference in distance each run. Also, the original implementation only starts its whole neighbor discovery after an original (self-generated/produced) routing announcement is received from a new neighbor. The AM is triggered on the first routing announcement received from that new neighbor, even if that announcement is a re-broadcast from another node in the new neighbor's network.

Chapter 6

Discussion

6.1 Modified Routing Announcements Vulnerability

The design was changed a bit during implementation, and with limited time some parts did not make it to the implementation These things needs to be addressed before the system can be used in a real world scenario. There are two attacks that the system are still vulnerable to, namely the *wormhole attack* and the *suppress replay attack*.

Because the routing announcements are unencrypted and the one-time-passwords are just appended after the messages, these packets can be altered using the following two attacks. In this section the two attacks are first explained, and then the section goes on to describe possible solutions for both attacks.

6.1.1 Wormhole Attack

In a wormhole attack, see Figure 2.4, an attacker does not need to know the keystream of a node in order to send that node's routing announcements. If you look at the figure from the background chapter you have a network of trusted nodes in green, and two malicious nodes in red.

Assume the two attackers, M1 and M2, want to disrupt the network topology by having node B believe node A (and vice versa) is a direct neighbor. Assuming the out-of-band wormhole in the figure is a faster route between node A and B than the "real" route through the network, M1 can simply forward the announcements from A to M2. Then M2 does the same, except he also needs to spoof his network and link layer addresses. When B receives the routing announcement from M2, he believes it is from A and that A is a new direct neighbor.

Now B will ask for A's keystream-material, and M2 and M1 will forward this request to A, which will dutifully reply with his keystream-material. Note that M1 needs to spoof B's addresses here, as well as forwarding B's routing announcements. When B receives the keystream through the wormhole, he is now able to verify the routing announcements also sent through the wormhole, and soon will this route take priority in his routing table as a strong direct link instead of going through other nodes in the network to reach A.

This attack only distorts the network topology, but notice that because the routing announcements' integrity are not protected, and the content is not encrypted, the attackers can possibly do greater damage by altering the content of the message. By altering the content, they could possibly generate fake re-broadcast announcements announcing untrusted nodes, possibly connecting a whole network of malicious nodes to this trusted network. This alteration is not actually a wormhole attack, but it is fair to say that the wormhole attack opens up to a variety of other attacks, such as alteration of packet, dropping of packets etc.

6.1.2 Suppress Replay Attack

Another attack which is possible due to the fact that the one time passwords are not "connected" to the routing announcements in any way is the suppress replay attack. Suppose an attacker is able to jam the signals for a very short time, only enough to distort the main payload of the routing announcement, while the one time password and its sequence number are kept intact.

The trusted receiver of this distorted packet will ignore it, because he will not understand the meaning of the packet. The attacker on the other hand, knows that the following uncorrupt data is a valid one time password and its sequence number from the sender's keystream.

Because the original recipient of this packet did not understand the destroyed packet, he will not know that this One-Time Password (OTP) is already used, and if the attacker wishes to create a false routing announcement he can now do this and append the OTP, which will be accepted by the original recipient. Note that also here the attacker needs to spoof the addresses of the original sender of the announcement which he partly jammed.

6.1.3 Possible Solution to the Suppress Replay Attack

A possible solution to the suppress replay attack, is to ensure message integrity in addition to the authentication. The basic idea would be to create and append a message digest of the announcement and an OTP from the keystream.

Using this idea, the OTP would never be sent visible, meaning at least one part of the routing announcement is a secret between the sender and the authorized receivers. The OTP has to be secret, or else an attacker could modify the message and create a new digest, whereas with a secret OTP the attacker would not be able to create an announcement which would give a verifiable digest value (collision). The message would therefore look like the following:

$Announcement|Hash(Announcement|OTP)|OTPSeqNum$

Note that the full length of the message digest (hash value) would not have to be appended to the message. A truncated version could be used, but when designing such a system one needs to calculate how long the truncated digest should be. The sliding window used to avoid replay attacks is only 64 bits long, and in the worst case (best case for adversary) there are only two legitimate nodes in the network. If this is the case, each node will send two announcements every second (BATMAN protocol) meaning the window of opportunity for the attacker is only 32 seconds. This means that the truncated digest should at least be of a size which is unfeasible for an attacker to find a digest collision within 32 seconds of brute forcing.

However, this requirement might even be to strict, because the attacker also has to find a collision which would produce valid values for the routing announcement - making the attackers job much more complex. This is true because the routing algorithms will drop any announcement containing values that does not “make any sense”.

Note also that even IF we suppose the attacker was able to find a collision for a valid announcement, the attacker would have to accomplish this continuously again and again, because the routing protocol (BATMAN) requires several valid announcements from a node before it updates its routes for that node. For each time the attacker fails at finding a valid and fake announcement the routing protocol at the receiving end figures this to be a bad or broken link, effectively killing the attack. It might be that the original two bytes, and its 65536 (2^16) possible combinations would be to difficult for an attacker to break.

6.1.4 Possible Solution to the Wormhole Attack

This attack vector is very difficult to to protect against. With the solution above, many attacks dependent on the wormhole attack are thwarted because of packet secrecy, integrity and authentication. However, the solution does not hinder an attacker from replaying the exact same packets through a wormhole, in order to alter the network topology. There has been a lot of research to find a good solution to this attack, but most solutions are aimed towards stationary networks and not Mobile Ad Hoc Networks (MANETs) [RH11].

One possible solution as pointed to by the article above is the use of “location aware guard node” and graph theory [PL07] [LPM$^+$05] to detect wormholes. The idea is that if you have special nodes spread out in the field at fixed points, where none of these nodes are within each other's transmitting range, one should never be a neighbor of more than one node at a time. If you receive direct packets from two nodes within a very short time frame, there might be indications that there's a wormhole in place replaying one of the special nodes' announcements.

6.2 Key Usage

In the fields of information security and cryptography it is not considered a good practice to use the same keys for different type of tasks such as encryption for confidentiality and signing for integrity and authenticity. It is argued that using only one key for different purposes brings a single point of failure to the design. While this is true, sometimes the benefits of having single key for these purposes are greater than the risk. In the proposed design the same public key pair is used to digitally sign keystream-material messages as well as when encrypting (with the recipient's public key) the symmetric keys in the same messages. In addition, the Service Proxys (SPs) use the same key to sign other nodes' Proxy Certificates (PCs).

The benefits of only having a singe public key pair for each node in the network is simplicity in the design. When it comes to the possible vulnerability if the one key is compromised, the lifetime of each key which is bound by the relatively short lifetime of PCs makes the window of opportunity for misuse of this key short and damage low.

6.3 Future Work - Extending the System Design

The specialization project [BG10] that preceded this thesis mentioned other important features the secure ad hoc network implementation should have. The things mentioned in this thesis' system design are actually implemented, but there are still much more that should be added if this system should be used in real life emergency situations.

In this section some important features which should be added, or at least studied, are mentioned. They do add more complexity to the system, so it is probably better to do a complete security and performance analysis on this thesis' proposed design before adding these features.

6.3.1 Initial Authentication with Long-Lived Public Key Certificates

One limiting factor in the system design is the need for an out-of-band initial authentication. With this limitation, every actor in the emergency scenario needs to manually verify his or her identity to the network management handled by the SP. With many actors, which would be typical in a large emergency situation like a natural disaster, this process might take up valuable time from the actual emergency work - which contradicts the whole meaning of setting up the MANET at the scene in the first place.

Now, if an actor has possession of a regular Long-Lived Public Key Certificate (LLPKC) which the network SP is able to verify, this should be allowed without the need of an out-of-band authentication. After the SP has verified the certificate, he can now issue the the actor a proxy certificate, signed by him so that all nodes in the network are able to verify the new node's identity.

The question of whether the SP is able to verify the LLPKC is not necessarily easy to answer. If the SP trusts the identity and knows the public key of the issuer of the actor's certificate, he is able to verify that this actor was trusted with this identity (and rights) at some time. However, it does not mean this is true anymore - the certificate might have been revoked.

In the absence of Internet access, Certificate Revocation Lists (CRLs) might not be available to the SP. If this is the case, an evaluation of what to do with the actor has to be done. Ideas that comes to mind might be to issue a very short lived PC, maybe with limited rights, to the actor's node so that it can start working now, but has to re-authenticate later when CRLs has been brought to the scene either out-of-band or by setting up Internet access. If no CRL is ever brought to the SPs attention, he might want to require an out-of-band authentication later.

Either way, as this does not have to happen in the same out-of-band fashion as the proposed design in this thesis requires, one could also allow the authentication to happen even if the actor is not a direct neighbor of the SP, but is connected through other nodes in the network. While in this implementation regular trusted nodes drop the announcements from new unauthenticated nodes, they could rather tunnel the announcement directly to the SP and have the authentication handshake go through them.

6.3.2 Network Merging

Due to R5 of Table 3.1 and for this implementation to be really useful for emergency services, which will consist of different types of actors and organizations, the implementation should support merging of multiple ad hoc networks. For example, if paramedics set up their own network, and firefighters arrive with their own network, co-operation between these networks should be supported.

Figure 6.1 (from the specialization project [BG10]) shows a possible scenario when a large emergency situation such as a natural disaster has happened. The figure portrays 4 organizational actors, i.e. police, firefighters, paramedics, and the military, which all run their own networks. Some of the networks co-operate through the use of gateways, such as between the police and the other actors, and the paramedics and firefighters co-operate by merging their networks.

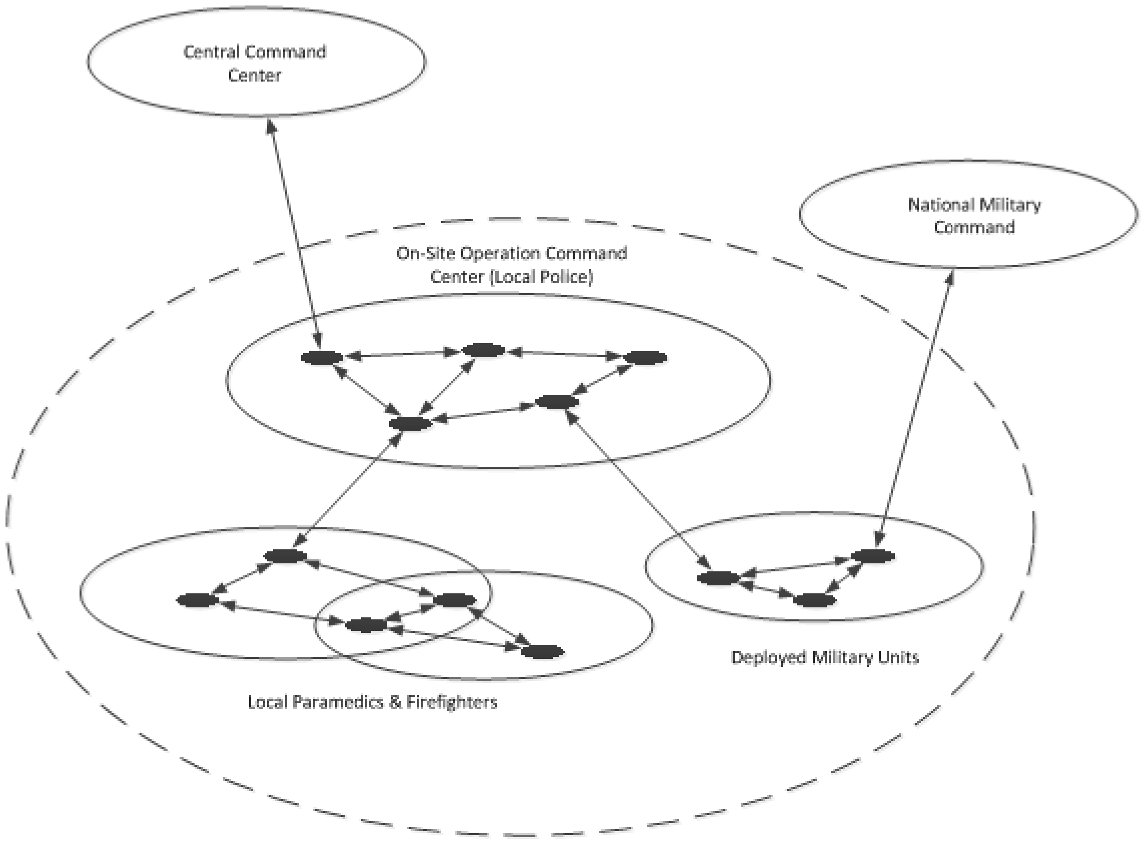

Central Command Center
National Military Command
On-Site Operation Command Center (Local Police)
Local Paramedics & Firefighters
Deployed Military Units

Figure 6.1: Scene from scenario with actors from multiple organizations.

Choosing whether to merge two networks or not is mainly a security issue. Some actors might have higher security concerns than others, e.g. military units does not trust the other actors to merge with their networks, but they might still want to communicate with them. This can for example be handled by having certain gateway nodes being allowed to communicate with nodes outside the network in a controlled manner.

Full Merging

A *full merging* means that two or more separately built and maintained networks completely merge so that each node in each of the networks receive all the routing announcements being sent within the networks. What this basically means is that the different networks become a single network.

The transition to from multiple to one network is not easy. Up to this point the assumption has been that there is only one management node called SP in the network. This will now change. With multiple networks merging into one there will be multiple SPs managing the access control of the network. In the next section having multiple SPs is discussed in greater detail.

There is one more major factor which needs to be addressed with full merging of networks, namely how access control is defined. One now needs to assume that the SPs either use the same policy languages to define what rights a node have been attributed with, or that they have a way of translating these policy languages so that they can be uniformly understood. For instance, in Section 3.7.1 three special attributes and their values were used as an example of how a policy might look like:

- Role - Node's role in the network
- Routing - Whether the node can partake in the routing
- Application - Whether the node has access to the application layer

Now, these attributes would seem to make sense in most applications of this system design, but some implementations might even have more attributes. Their values might also be different, because different networks might have different needs, for instance the 'role' attribute might have different values depending on what kind of elements are present in the network. For instance, one network might have the "location aware guard nodes" mentioned earlier in the network for wormhole protection, while other network do not. With two such different networks, full merging might not be feasible, but with smaller differences the obstacles might be manageable.

Limited Merging

Limited merging is a completely different technique than the one described above, and probably easier to implement. With limited merging of two or more networks, the networks stay autonomous with their own (possible) hierarchy, set of rules, management nodes, and so on, while at the same time allowing the two (or more) networks to collaborate. What constitutes the merging is that at least one node in each network is set up with gateway capabilities, which can broadcast the existence of another network to the nodes inside its own network.

Rules on the application layer must necessarily still be looked at, as they should work over the different networks, but routing rules and other access controls can stay hidden between the networks.

Internet Gateway

Similar to limited merging, a node in a network may be given gateway capabilities to announce Internet access to the network. Because all nodes belong to the same MANET and the management is consistent within this one network, this capability should be relatively easy to implement. The only thing the reader should notice is that the policy field within the proxy certificates of the nodes can now be used to declare whether the node should have Internet access or not by adding an *Internet attribute* to the policy.

6.3.3 Multiple Service Proxies

Whenever full merging of two secure and managed MANETs happen as described above two *or more* (!) SPs will end up in the same network (the emphasis on more is used because one network might have more than one SP). This would be a challenging task to overcome, for these networks are inherently flat, or non-hierarchical. Decision making in the network after this point would be difficult if one network (or specifically one SP) sees itself as more important than the other.

It is probably better to say that for two networks to fully merge, they should see themselves as equal counterpart where no network has more rights than the other, and therefore the SPs have the same rights as each other. This means that nodes issued by one SP, or said differently one SP's children, are equally trusted by all other nodes in the network, depending on the restrictions set in their PCs of course.

If all the SPs in the network are considered "equal", then they are all equally trusted to issue new nodes and introduce them to the network. The major requirement missing for this scheme to happen is that all nodes in the network now needs to know the public keys for all SPs.

It is likely that not all nodes in the network will ever become a direct neighbor and therefore not receive the other SPs' Proxy Certificate 0s (PC0s), but they will meet nodes signed by those SPs. A way around this problem is to have each SP to once in a while broadcast a digitally signed list to all its children nodes containing the identities (unique subject names) and public keys for all other trusted SPs in the network. Because this list is signed by a known SP, you are able to verify the authenticity of this list and therefore implicitly verify and trust all nodes issued with PCs from the other SPs.

6.4 Experience with OpenSSL

The experience with OpenSSL has been a challenging one. There is little documentation available, and only on some segments of the library. Many functions being used in the implementation of this thesis are undocumented, or at least not in a freely available documentation. Not to be confused, you can find most OpenSSL functions in their documentation, but most of those functions are not explained, many do not explain output and input to functions, and many functions have unclear names making it a game of “guessing” which functions to use.

Most of the implementation has been understood by looking at a few available examples online at different sites, some demos within the OpenSSL libraries, and by asking the online community. For reference, I would personally recommend the community at *Stack Overflow*[1] rather than the OpenSSL mailing lists[2] [3] which have not answered any questions I raised.

When I’ve had enough time I’ve documented the OpenSSL functions used in my implementation, which can hopefully help someone planning to further implement on this design, or other designs with similar functionality.

During the implementation and before completing the keystream sharing, ECC was used instead of RSA for signing the certificates. In order to use ECC for the keystream sharing, ECDH or ECIES had to be used. This however, was not an easy task using OpenSSL, and the author had to change the public keys to RSA in order to complete the keystream sharing in due time before turnging in this thesis. One can easily argue that ECC should be used rather than RSA because of key sizes, making much sense in MANETs.

6.5 Out-Of-Band Authentication

If an authentication scheme use some other medium to authenticate a user than the medium for which the authenticated identities are supposed to use later, you have an out-of-band authentication. For instance, when two computers interact over the Internet, and the authentication scheme relies on some secret information being transferred via e.g. SMS on the phone network, then this secret information is being shared on an out-of-band authentication.

For the purpose of this thesis, a few out-of-band authentication schemes could be handy, such as sharing public keys (or just their fingerprints) verbally between the human actors, or using USB sticks. There is no requirement of sharing the public keys however. If one can assume that you know with certainty that a communication

[1] http://stackoverflow.com
[2] openssl-dev@openssl.org
[3] openssl-users@openssl.org

is from the alleged identity, you can instantly grant this identity access and issue it a PC. This can be true if the SP is plugged into one end of an ethernet cable, and the other end of the cable is plugged into the actors machine. Then you know this machine belongs to the actor and can be directly verified.

Another part of the out-of-band authentication can be that an actor has a non-digital authentication token such as a passport, or a work related identity badge issued by the emergency organization employing this actor. The SP might require this person to show him this identity badge before being allowed into the network. In other words, there are countless of different out-of-band authentication schemes, and they will not be discussed here further.

Chapter 7

Conclusion

With this thesis I have shown that the problem of authentication in Mobile Ad Hoc Networks can be managed using a special type of public key certificates called proxy certificates, along with the use of one-time passwords. In this thesis a system design for handling the authentication in a MANET is proposed along with its implementation for UNIX systems. I have shown that a central node is only required for initial authentication, while the subsequent operation of the MANET have no such requirement due to the use of proxy certificates and a trust scheme.

Prior to this thesis, a comprehensive study of related works for authentication and security in ad hoc networks was carried out during a specialization project in co-operation with Anne Gabrielle Bowitz. During this study the main design ideas were formed, but changes were made during this thesis to account for authentication of nodes without a verifiable authentication token, and also during the implementation when some specific goals became more clear.

An emergency scenario was constructed and used to create system design requirements. With the exception of one requirement, which has been deemed further work, all requirements have been accounted for and handled in the proposed system design.

The implementation of the system design was achieved by extending an existing and popular ad hoc routing protocol called BATMAN. The extension should also be portable to other network-layer pro-active routing protocols, with only minor modifications.

Furthermore, the performance of the implemented system design was compared to the original routing protocol implementation. These tests indicates that the extension do not add much delay, but rather a small and constant delay. The tests are, however not fully definite and they should probably be runned again with a larger set of test-machines and with far more trials. Only this way can a linear time delay be disproven. If further testing should prove a constant delay, there would be no obvious performance issues with the proposed design.

Some tasks are left as future work. As noted in the discussion chapter, both wormhole attacks and suppress replay attacks might have to be countered. With an powerful adversary both attacks could be successful on the proposed design, however two possible solutions (one more or less straight forward) were discussed. These proposals are left as my suggestions to the science community for further work. A peer review of the security assumptions and choices in the proposed design is also left as further work.

References

[BG10] A.G. Bowitz and E.G. Graarud. Developing a Secure Ad Hoc Network Implementation. Technical report, NTNU ITEM, 2010. https://github.com/espengra/secure-ad-hoc-network-doc/raw/master/share/project.pdf.

[Bro10] L. Brown. Using Proxy Certificates for Mobile and Other Authentication Needs, 2010. http://www.unsw.adfa.edu.au/~lpb/papers/ssp10/xpc-mob-10a.html.

[Chr11] J. Chroboczek. The Babel Routing Protocol. Technical report, RFC Editor, 2011. http://www.rfc-editor.org/rfc/rfc6126.txt.

[CJ10] Thomas Heide Clausen and Philippe Jacquet. Optimized Link State Routing Protocol (OLSR). *Network Working Group*, Last accessed December 19, 2010. http://tools.ietf.org/html/rfc3626.

[CJA+03] T. Clausen, P. Jacquet, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot. Optimized Link State Routing Protocol (OLSR), 2003. Network Working Group Network Working Group.

[DD11] H. Dey and R. Datta. A threshold cryptography based authentication scheme for mobile ad-hoc network. In Natarajan Meghanathan, Brajesh Kumar Kaushik, and Dhinaharan Nagamalai, editors, *Advances in Networks and Communications*, volume 132 of *Communications in Computer and Information Science*, pages 400–409. Springer Berlin Heidelberg, 2011.

[DLRS01] B. Dahill, B.N. Levine, E. Royer, and C. Shields. A secure routing protocol for ad hoc networks. *Electrical Engineering and Computer Science, University of Michigan, Tech. Rep. UM-CS-2001-037*, 2001.

[DOG+] S.K. Dhurandher, M.S. Obaidat, D. Gupta, N. Gupta, and A. Asthana. Network layer based secure routing protocol for wireless ad hoc sensor networks in urban environments. In *Wireless Information Networks and Systems (WINSYS), Proceedings of the 2010 International Conference on*, pages 1–6. IEEE.

[FKTT98] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *Proceedings of the 5th ACM conference on Computer and communications security*, pages 83–92. ACM, 1998.

[GBS10] P. Goyal, S. Batra, and A. Singh. A literature review of security attack in mobile ad-hoc networks. *International Journal of Computer Applications IJCA*, 9(12):24–28, 2010.

[Hal94] N.M. Haller. The s/key (tm) one-time password system. In *Symposium on Network and Distributed System Security*, pages 151–157. Citeseer, 1994.

[He02] G. He. Destination-sequenced distance vector (dsdv) protocol. *Networking Laboratory, Helsinki University of Technology*, 2002.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. *Network Working Group*, January 1999. http://www.ietf.org/rfc/rfc2459.txt.

[HM03] A.R. Hevner and S.T. March. The information systems research cycle. *Computer*, 36(11):111–113, 2003.

[HPJ05] Y.C. Hu, A. Perrig, and D.B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. *Wireless Networks*, 11(1):21–38, 2005.

[HSBW11] S.A. Hosseini Seno, R. Budiarto, and T-C. Wan. A Secure Mobile Ad hoc Network Based on Distributed Certificate Authority. *Arabian Journal for Science and Engineering*, pages 245–257, 2011.

[Lin93] J. Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. *Network Working Group*, 1993. http://www.ietf.org/rfc/rfc1421.txt.

[LPM$^+$05] L. Lazos, R. Poovendran, C. Meadows, P. Syverson, and LW Chang. Preventing wormhole attacks on wireless ad hoc networks: a graph theoretic approach. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 2, pages 1193–1199. IEEE, 2005.

[MDK10] D. Murray, M. Dixon, and T. Koziniec. An experimental comparison of routing protocols in multi hop ad hoc networks. In *Telecommunication Networks and Applications Conference (ATNAC), 2010 Australasian*, pages 159 –164, 31 2010-nov. 3 2010.

[Mes10] Open Mesh. *Why starting B.A.T.M.A.N.?* *open-mesh.org*, Last accessed december 19, 2010. http://www.open-mesh.org/wiki/why-starting-batman.

[NALW10] A. Neumann, C. Aichele, M. Lindner, and S. Wunderlich. Better Approach To Ad-Hoc Networking (B.A.T.M.A.N) draft-wunderlich-openmesh-manet-routing-00. *Network Working Group*, Last accessed December 19, 2010. http://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00.

[NJT09] Å.A. Nyre, M.G. Jaatun, and I.A. Tøndel. A secure MANET routing protocol for first responders. In *Security and Communication Networks (IWSCN), 2009 Proceedings of the 1st International Workshop on*, pages 1–7. IEEE, 2009.

[PBRD03] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing, 2003.

[PD07] L.L. Peterson and B.S. Davie. *Computer networks: a systems approach*. Morgan Kaufmann Pub, 2007.

[Per08] C.E. Perkins. *Ad hoc networking*. Addison-Wesley Professional, 2008.

[PH02] P. Papadimitratos and Z.J. Haas. Secure routing for mobile ad hoc networks. In *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, volume 31. Citeseer, 2002.

[PL07] R. Poovendran and L. Lazos. A graph theoretic framework for preventing the wormhole attack in wireless ad hoc networks. *Wireless Networks*, 13(1):27–59, 2007.

[PM04] A.A. Pirzada and C. McDonald. Establishing trust in pure ad-hoc networks. In *Proceedings of the 27th Australasian conference on Computer science - Volume 26*, ACSC '04, pages 47–54, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[RH11] M.N.S. Raote and M.K.N. Hande. Approaches towards mitigating wormhole attack in wireless ad-hoc network. *IJAEST*, 2(2):170–173, 2011.

[SDL$^+$02] K. Sanzgiri, B. Dahill, B.N. Levine, C. Shields, and E.M. Belding-Royer. A secure routing protocol for ad hoc networks. *Network Protocols, IEEE International Conference on*, 0:78, 2002.

[SLD$^+$05] K. Sanzgiri, D. LaFlamme, B. Dahill, B.N. Levine, C. Shields, and E.M. Belding-Royer. Authenticated routing for ad hoc networks. *Selected Areas in Communications, IEEE Journal on*, 23(3):598–610, 2005.

[Sva08] I. Svagard. Information security for field workers in crisis situations. Technical report, SINTEF ICT, http://www.oasis-fp6.org/documents/OASIS_SP24_DDD_253_security_SIN_1_0_pub.pdf, 2008.

[TET$^+$04] S. Tuecke, D. Engert, M. Thompson, L. Pearlman, and V. Welch. Internet X.509 Public Key Infrastructure Proxy Certificate Profile. Technical report, RFC Editor, 2004. http://tools.ietf.org/html/rfc3820.

[TJN09] I.A. Tøndel, M.G. Jaatun, and A.A. Nyre. Security requirements for MANETs used in emergency and rescue operations. In *Security and Communication Networks (IWSCN), 2009 Proceedings of the 1st International Workshop on*, pages 1–7. IEEE, 2009.

[VMC02] J. Viega, M. Messier, and P. Chandra. *Network security with OpenSSL*. O'Reilly Media, 2002.

[WSK06] E. Winjum, P. Spilling, and Ø. Kure. *Ad Hoc networks used in emergency networks : the Trust Metric Routing approach*. FFI Rapport, 2006.

[Zim80] H. Zimmermann. OSI reference model–The ISO model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, 1980.

[Zim95] P.R. Zimmermann. The official PGP user's guide. 1995.

Appendix A

Source Code

A.1 Complete Source Code

The source code is released in two different ways. The version used to produce the test results in this thesis is released in a zipped package at the following address:

https://github.com/espengra/secure-ad-hoc-network-doc/raw/master/share/secure-ad-hoc-network.zip

For the latest updated version there is a Git repository where you can download or fork your own branch of the source code from the following address:

https://github.com/espengra/secure-ad-hoc-network

A.2 Code Snippets

In this section all the code snippets referred to from Chapter 4 are shown. They might not reflect the source code perfectly, as some lines in between and re-organization has been done to only include the most important steps. However, all the values are the exact same as their counterparts in the source code.

A.2.1 AM Sockets Setup

```
int32_t *recvsock, *sendsock;
addrinfo hints, *res;

/* Set family information */
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_flags = AI_PASSIVE;
hints.ai_protocol = IPPROTO_UDP;

/* Puts the port-info inside a addrinfo data structure */
getaddrinfo(NULL, port, &hints, &res);

/* Assign file descriptor for sockets */
*recvsock = socket(PF_INET, SOCK_DGRAM, 0)
*sendsock = socket(PF_INET, SOCK_DGRAM, 0)

/* Binds the sockets to the network interface */
setsockopt(*recvsock, SOL_SOCKET, SO_BINDTODEVICE, interface,
    strlen(interface) + 1)
setsockopt(*sendsock, SOL_SOCKET, SO_BINDTODEVICE, interface,
    strlen(interface) + 1)

/* Binds receive socket to the port (rest of the address is
    empty/null) */
bind(*recvsock, res->ai_addr, res->ai_addrlen);

/* Allow the send socket to send broadcast messages */
int broadcast_val = 1;
setsockopt(*sendsock, SOL_SOCKET, SO_BROADCAST, &broadcast_val,
    sizeof int)

/* Set the send socket to non-blocking */
fcntl(*sendsock, F_SETFL, O_NONBLOCK);
```

A.2.2 Proxy Certificate Extension

Remember the author did not get the original proxyCertInfoExtension to work, so it was changed with a netscape comment.

```
STACK_OF(X509_EXTENSION) *exts = sk_X509_EXTENSION_new_null();
openssl_cert_add_ext_req(exts, NID_netscape_comment, "critical,
    myProxyCertInfoExtension:0,1");

X509_REQ_add_extensions(x, exts);
sk_X509_EXTENSION_pop_free(exts, X509_EXTENSION_free);
```

The first digit with a 0 in the comment tells the role the node requests is only regular 'authenticated'. If this was 1 it would mean the node was a SP. The last digit is used for routing rights. The 1 in the last digit means the node wishes full routing rights, whereas a 0 would mean limited routing rights.

A.2.3 Setting Subject Name in PC

```
X509_NAME *name, *req_name, *issuer_name;
req_name = X509_REQ_get_subject_name(req)
issuer_name = X509_get_subject_name(*pc0p)
name = X509_NAME_dup(issuer_name)
req_name_entry = X509_NAME_get_entry(req_name,0);
X509_NAME_add_entry(name, req_name_entry, X509_NAME_entry_count(
    name), 0);
X509_set_subject_name(cert, name)
```

A.2.4 Adding Trusted Node to AL

```
void al_add(uint32_t addr, uint16_t id, role_type role, unsigned
    char *subject_name, EVP_PKEY *key) {

  authenticated_list[num_auth_nodes] = malloc(sizeof(
      trusted_node));
  authenticated_list[num_auth_nodes]->addr = addr;
  authenticated_list[num_auth_nodes]->id = id;
  authenticated_list[num_auth_nodes]->role = role;
  authenticated_list[num_auth_nodes]->name = malloc(
      FULL_SUB_NM_SZ);
  memset(authenticated_list[num_auth_nodes]->name, 0,
      FULL_SUB_NM_SZ);

  if(strlen((char *)subject_name)>FULL_SUB_NM_SZ)
    memcpy(authenticated_list[num_auth_nodes]->name,
        subject_name, FULL_SUB_NM_SZ);
  else
    memcpy(authenticated_list[num_auth_nodes]->name,
        subject_name, strlen((char *)subject_name));

  authenticated_list[num_auth_nodes]->pub_key = openssl_key_copy
      (key);

  if(id != my_id) {
    EVP_PKEY_free(key);
  }

  num_auth_nodes++;

}
```

A.2.5 Adding Trusted Neighbor to NL

```
void neigh_list_add(uint32_t addr, uint16_t id, unsigned char *
   mac_value) {

  int i;
  for(i=0; i<num_trusted_neigh; i++) {
    if(id == neigh_list[i]->id) {

      if(addr == neigh_list[i]->addr) {

        if(neigh_list[i]->mac != NULL)
          free(neigh_list[i]->mac);

        neigh_list[i]->mac = mac_value;
        neigh_list[i]->window = 0;
        neigh_list[i]->last_seq_num = 0;
        neigh_list[i]->last_rcvd_time = time (NULL);
        neigh_list[i]->num_keystream_fails = 0;

      } else {

        if (mac_value != NULL)
          free(mac_value);

        neig_list_remove(i);

      }

      break;

    }

  }

  if(i==num_trusted_neigh) {

    neigh_list[num_trusted_neigh] = malloc(sizeof(trusted_neigh)
       );
    neigh_list[num_trusted_neigh]->addr = addr;
    neigh_list[num_trusted_neigh]->id = id;
    neigh_list[num_trusted_neigh]->mac = mac_value;
    neigh_list[i]->window = 0;
    neigh_list[num_trusted_neigh]->last_seq_num = 0;
    neigh_list[num_trusted_neigh]->last_rcvd_time = time (NULL);
    neigh_list[num_trusted_neigh]->num_keystream_fails = 0;
    num_trusted_neigh++;
  }

}
```

A.2.6 Removing Trusted Neighbor to NL

```
int neig_list_remove(int pos) {

  /* First check whether this node exists at all (sanity check)
     */
  if(neigh_list[pos] == NULL) {
    return 0;
  }

  /* Check whether keystream exists, remove if so! */
  if(neigh_list[pos]->mac != NULL)
    free(neigh_list[pos]->mac);

  /* Free up neighbor in memory */
  free(neigh_list[pos]);

  /* Re-arrange Neighbor List to avoid scarce population */
  int i;
  for(i=pos+1; i<num_trusted_neigh; i++) {
    neigh_list[i-1] = neigh_list[i];
  }

  /* Finally, number of trusted neighbors has shrunk :) */
  num_trusted_neigh--;

  return 1;
}
```

A.2.7 Generate Ephemeral Key

```
void openssl_key_generate(EVP_CIPHER_CTX *aes_master, int
   key_count, unsigned char **keyp) {

  unsigned char *ret;
  int i, tmp, ol;

  if(keyp == NULL || *keyp == NULL) {
    ret = malloc(EVP_CIPHER_CTX_block_size(aes_master));
  } else {
    memset(*keyp, 0, EVP_CIPHER_CTX_block_size(aes_master));
    ret = *keyp;
  }

  ol = 0;

  /* Create plaintext from key_count - each new key will be
     cipher of i=1,2,3... */
  unsigned char *plaintext = malloc(sizeof(key_count));
  memset(plaintext, 0, sizeof(plaintext));
  *plaintext = (unsigned char)key_count;
  int len = strlen((char *)plaintext)+1;

  EVP_EncryptUpdate(aes_master, ret, &tmp, plaintext, len);
  ol += tmp;
  //Remove padding, not wanted for key!
  EVP_EncryptFinal(aes_master, ret, &tmp);

  free(plaintext);
  *keyp = ret;

}
```

A.2.8 Generate Keystream

```
/* Generate Keystream from Nonce */

if(*key_count>1)
  free(auth_value);

int rand_len = RAND_LEN;
auth_value = malloc(rand_len*10+10);
auth_value_len = 0;

for(i=0; i<10; i++) {

  /* Do encryption */
  EVP_CIPHER_CTX current_ctx;
  EVP_EncryptInit(&current_ctx, EVP_aes_128_cbc(), current_key,
     current_iv);
  unsigned char *tmp = openssl_aes_encrypt(&current_ctx,
     current_rand, &value_len);
  EVP_CIPHER_CTX_cleanup(&current_ctx);

  /* Place ciphertext in keystream */
  int auth_pos = auth_value_len;
  auth_value_len += value_len;
  memcpy(auth_value+auth_pos, tmp, value_len);

  /* Change to new IV */
  memcpy(current_iv, tmp, AES_IV_SIZE);

  /* Alter the Nonce before next encryption */
  int j;
  for(j=0;j<rand_len/10; j++) {
    current_rand[j+(i*(rand_len/10))] = ( (current_rand[j+(i*(
       rand_len/10))]) ^ i );
  }

  free(tmp);
  value_len = RAND_LEN;

}
```

A.2.9 Extension in BATMAN Class

```
/******************* Begin Authentication Module Extension
   *******************/

/*
 * If the daemon is not authenticated, or it receives an
    authentication
 * token which it does not recognize, the authentication
    procedure in the
 * Authentication Module is called. No packets received when
    authenticating
 * will be processed.
 */

if(num_trusted_neigh) {
  for(neigh_counter = 0; neigh_counter < num_trusted_neigh;
     neigh_counter++) {
    if(neigh_list[neigh_counter]->addr == neigh) {
      break;
    }
  }
}

if(neigh_counter == num_trusted_neigh) {

  if(my_role == SP && my_state == READY) {
    new_neighbor = neigh;
  }

  if(my_role == AUTHENTICATED && my_state ==  READY) {
    /* Check to see whether the other node is AUTHENTICATED */
    if(memcmp(&(bat_packet->auth), empty_check, 2) != 0)
      new_neighbor = neigh;
  }

  goto send_packets;
}

if(neigh_list[neigh_counter]->mac == NULL)
  goto send_packets;

if(memcmp(neigh_list[neigh_counter]->mac+(bat_packet->auth_seqno
    *2), bat_packet->auth, 2) != 0) {

  printf("MAC Extract did not match!\n");

  if(my_state == READY) {

    neigh_list[neigh_counter]->num_keystream_fails ++;

    /* Keystream is consequently fail, ergo need to handshake a
       new one */
    if(neigh_list[neigh_counter]->num_keystream_fails > 20) {
      my_state = WAIT_FOR_REQ_SIG;
      new_neighbor = neigh;
      neigh_list[neigh_counter]->num_keystream_fails = 0;
    }

  }

  goto send_packets;
}

/* Check whether the packet is new and not a replayed packet */
if(!tool_sliding_window(bat_packet->auth_seqno, neigh_list[
    neigh_counter]->id))
  goto send_packets;


/* Everything seems fine, reset failcounter if more than 0 */
if(neigh_list[neigh_counter]->num_keystream_fails != 0)
  neigh_list[neigh_counter]->num_keystream_fails = 0;

/******************** End Authentication Module Extension
    ********************/
```

A.2.10 Extension in SCHEDULE Class

```
/* Begin Authentication Module Extension */

/* Add Signature Extract to OGM */
if(pthread_mutex_trylock(&auth_lock) == 0) {

  if(auth_value != NULL) {

    memcpy(bat_packet->auth, auth_value+2*auth_seq_num, 2);
    bat_packet->auth_seqno = auth_seq_num;
    auth_seq_num ++;

  }

  pthread_mutex_unlock(&auth_lock);
}

/* End Authentication Module Extension */
```

Appendix B

Lab Setup

The computers used in the lab was setup with the following hardware:

- Intel Core 2 Duo 2.83 GHz processor
- 4 GB memory
- Atheros AR5413 802.11abg NIC

Further, they are setup with Ubuntu 10.4 (Linux Kernel 2.6.32-25-generic-pae) and ath5k drivers for the wireless interfaces. The network interface is configured as follows:

```
/etc/network/interfaces

auto lo
iface lo inet loopback

auto wlan0
iface wlan0 inet static
address 10.0.0.X
netmask 255.255.255.0
pre-up ifconfig wlan0 down
pre-up ifconfig wlan0 hw ether XX:XX:XX:XX:XX:XX
pre-up iwconfig wlan0 mode ad-hoc essid BATMAN channel 3

auto unicast
iface unicast inet static
address 10.0.0.X
netmask 255.255.255.0
pre-up brctl addbr unicast
pre-up brctl addif unicast wlan0
pre-down ifconfig unicast down
```

```
post-down brctl delif unicast wlan0
post-down brctl delbr unicast
```

To install batmand, run the following as root user:

```
make
make install
make clean
```

To run the batman daemon on a regular test node, use the following command:

```
batmand --role authenticated -d 4 wlan0
```

To run the batman daemin on the SP node, use the following command:

```
batmand --role sp -d 4 wlan0
```

Appendix C

Test Results

C.1 Numerical Results

This sections presents the actual numerical results the graphs in Chapter 5, and their means and variances.

C.1.1 Test I - Original BATMAN

#	First OGM Received (s)	First Route Added (s)	Last Route Added (s)	Time to Add One (s)	Time To Add Both(s)
1	0.35	1.42	1.44	1.07	1.09
2	0.41	1.35	1.65	0.94	1.24
3	0.18	1.22	3.52	1.04	3.34
4	0.75	1.77	1.82	1.02	1.07
5	0.18	1.33	2.24	1.15	2.06
6	0.02	1.94	2.12	1.92	2.10
7	0.01	2.05	2.93	2.04	2.92
8	0.13	1.10	1.51	0.97	1.38
9	0.66	1.68	1.72	1.02	1.06
10	0.01	1.98	3.43	1.97	3.42

Table C.1: Test I using Original BATMAN.

Mean time to add first route: 1.31 seconds
Mean time to add both routes: 1.97 seconds

Standard deviation in time adding first route: 0,44 seconds
Standard deviation in time adding both routes: 0,91 seconds

C.1.2 Test I - Modified BATMAN

#	**First OGM Received (s)**	**First Route Added (s)**	**Last Route Added (s)**	**Time to Add One (s)**	**Time To Add Both(s)**
1	0.03	3.17	3.45	3.14	3.42
2	0.33	3.89	4.10	3.56	3.77
3	0.69	3.69	3.98	3.00	3.29
4	0.76	3.81	3.96	3.05	3.20
5	0.27	3.29	3.58	3.02	3.31
6	0.66	3.61	4.15	2.95	3.49
7	0.31	3.56	4.05	3.25	3.74
8	0.22	3.20	4.23	2.98	4.01
9	0.36	3.41	3.99	3.05	3.63
10	0.64	3.54	3.59	2.90	2.95

Table C.2: Test I using Modified BATMAN.

Mean time to add first route: 3.09 seconds
Mean time to add both routes: 3.48 seconds

Standard deviation in time adding first route: 0,18 seconds
Standard deviation in time adding both routes: 0,30 seconds

C.1.3 Test II

#	**Original Convergence Time (s)**	**Modified Convergence Time (s)**
1	5.75	2.05
2	4.97	4.64
3	6.62	4.04
4	4.60	1.72
5	4.64	3.35
6	6.34	3.10
7	5.65	12.76
8	6.06	10.18
9	6.16	12.01
10	3.72	6.60

Table C.3: Test II using both BATMAN versions.

Mean convergence time using original BATMAN: 5.45 seconds
Mean convergence time using modified BATMAN: 6.05 seconds

Standard deviation convergence time using original BATMAN: 0.88 seconds
Standard deviation convergence time using modified BATMAN: 3.93 seconds

C.2 Logs

All of the logs produced from the tests can be found at the following address:

https://github.com/espengra/secure-ad-hoc-network-doc/raw/master/share/results.zip

Note that the logs are in the BATMAN “-v 4” format. For explanation see the man pages for batmand.

Appendix D

Scientific Paper

Based on the work behind this and Anne Gabrielle Bowitz' thesis, we have in collaboration with our supervisors Martin Gilje Jaatun and Dr. Lawrie Brown produced a scientific paper for publishing. At the moment of this writing the thesis has been submitted for approval at one international scientific conference.

The submitted copy is appended at the following page. This might be revised after the submission of this thesis, and the updated version can be found at the following address:

https://github.com/espengra/secure-ad-hoc-network-doc/raw/master/share/paper.pdf

BatCave: Adding Security to the BATMAN Protocol

Blind review

***Abstract*—The Better Approach To Mobile Ad-hoc Networking (BATMAN) protocol is intended as a replacement for protocols such as OLSR, but just like most such efforts, BATMAN has no built-in security features. In this paper we describe security extensions to BATMAN that control network participation and prevent unauthorized nodes from influencing network routing.**

I. Introduction

This work developed from a perceived need to implement a secure adhoc network that might be used in emergency services, disaster assistance, and military applications. Such a network needs to be established quickly, and without the need for existing fixed infrastructure. However it also requires controls to limit access to the network, in order to protect it from intruders or unwanted bystanders. We propose extensions to a suitable adhoc network routing protocol, BATMAN, so that routing advertisements will only be accepted from authorised stations on the network. We propose the use of proxy certificates, which each client wishing to access the network will generate, and which are signed by one of the suitably authorised stations tasked with creating and manageing the network. We assume these stations will be located with suitable emergency services command units that the network is being created to support.

The remainder of this paper is structured as follows:

II. Related Work on Adhoc Network Security

Our proposals evolved from work on developing a secure restricted ad-hoc network for use by emergency services or disaster response personnel [1], [2]. In such a network, access must be managed, but be provided for members of multiple authorities which might not have online access to verify their identity. They focused on the design and implementation of the needed extensions to the OLSR adhoc network routing protocol. However they only made a brief mention of the use of a public-key infrastructure to identify mobile clients and to authorise their access to some restricted ad-hoc network. They suggested that clients in a region would be pre-configured with certificates that could be used to automatically grant them access. They also noted that there needs to be some means of granting access to mobile devices that are not known, for personnal from out of region or from other services without peering arrangements. They suggested that such devices can be issued short-lived certificates, with limited rights, to grant them access. However details of this were left mostly unspecified.

In other related work, short-lived X.509 certificates were proposed as a suitable mobile authentication method for low power or otherwise resource limited devices [3], [4]. The main reasons they gave for choosing such certificates, which are "conventional" X.509 certificates but with a much shorter lifetime of hours to days, include a desire to avoid the cost and overhead of checking a Certificate Revocation List (CRL) or otherwise handling detection of revoked certificates. It was also to allow the use of less computationally intensive algorithms and key sizes than may be required in "conventional" X.509 certificates with lifetimes, and hence need for sufficient strength against attack, over periods of months to years.

III. Addressing Limitiations in the Existing Work

Our proposed adhoc network security extensions address some issues with the prior work noted above. First was the choice of adhoc network routing protocol to modify. Although OLSR is an Internet standard, several papers have suggested thats its performance in practical trials is less than desired [5], [6]. Of the other protocols tested, it appears that BATMAN provided the best overall performance. We present further details on this choice in the next section.

Next was the choice of types of certificates to use to manage controlled admission to the network. The existing proposals involve using a mix of conventional and short-lived certificates, with the latter being generated in the field as required to support admission of stations without existing, verifiable, conventional certificates. However this means the stations issuing these need to support some certificate authority (CA) functionality, and have CA certificates available to sign these newly created certificates (short-lived or otherwise). Normal client stations would not normally have these.

We propose instead the use of proxy certificates, which are X.509 certificates with specific proxy extensions, that are signed either by another, conventional client certificate, or by a proxy certificate (PC), as we detail later in section VI-A. Hence any client station can potentially act a certificate issuer, able to grant access to other stations. The problem then becomes one of distributing knowledge of which stations have that authority, which we address as part of our protocol extensions. Note with our proposed use of proxy certificates, they become an access token or capability used to gain access to a service, in this case the adhoc network. This is very much the opposite sense to current use of these certificates, which are used by clients to delegate some of their access rights to a server, particularly in the grid computing domain [7].

Another problem not explicitly addressed in the previous work, is just what controls or restrictions were placed on the process of issuing certificates to grant access to the network. They identify the need to support differing categories of stations needing access. Some may be automatically recognized and trusted because they possess a conventional client certificate issued by a CA known to the proxy issuing client, most likely because both stations belong to the same service or administrative structure. In this case it would be reasonable to automatically issue the proxy certificate and grant network access without any human intervention. Other clients may not be immediately recognized, since they belong to other services, are volunteers, or just not previous known. In such cases it would seem reasonable to require manual verification that the client should be granted access before issuing a proxy certificate to them.

A further advantage in the use of proxy certificates is that they support the specification of restrictions on their use. We propose using this mechanism to assign different rights to different classes of clients. This could be used to indicate which clients are delegated the right to also issue proxy certificates granting access to other stations to the existing network. It also could be used to indicate that some stations should only be end-systems, and not used to relay traffic. Since X.509 certificates are widely recognized, it would also be possible to use the issued proxy certificates to authorise and authenticate the client's use of specific upper-layer applications.

IV. B.A.T.M.A.N.

BATMAN [8] ("Better Approach To Mobile Ad hoc Networking") is an increasingly popular routing protocol for wireless ad hoc networks, which was developed with an aim to replace the Optimized Link State Routing Protocol (OLSR) [9]. OLSR is a pro-active routing protocol, which means that participating nodes regularly exchange routing information with each other. According to the BATMAN developers, the problem with OLSR is that every node in the network calculates the whole routing path, which is a complex way to do it. Not only is it difficult to make sure all nodes have the same information at the same time, it also needs (relatively) much storage and computation time. If nodes sit on different routing information this concept leads to routing loops and heavy route flapping. The result is many patches to the protocol that defies the protocol standard in order to make it more suitable [9].

In BATMAN, each node should only know the next hop, i.e., the link-local neighbor that is the path between itself and the destination. BATMAN calculates the optimal route, i.e. the next jump, by comparing the number of routing messages it has received from each node and who was the last sender.

The routing messages sent in BATMAN are called OGM. Figure 1 shows the packet format with all header fields. The OGM format has changed since the BATMAN draft [8] was published, but there is no official publication with the new packet format as of yet. The packet format found in the RFC draft belongs to the older version III of the BATMAN algorithm. The algorithm used in this paper is version IV.

The real workhorse of the packet is the "Originator Address" field which carries a host address of the node 'A' that broadcasted the OGM. When a node 'B' receives this message it checks if the originator address and source address of the IP header are the same - if so the two nodes are direct neighbors. B then forwards the OGM only changing the "TTL" and "Previous Sender" fields. All OGM inside the BATMAN network are broadcasted and rebroadcasted until the TTL has dropped to zero, or until they receive an OGM they have previously sent themselves.

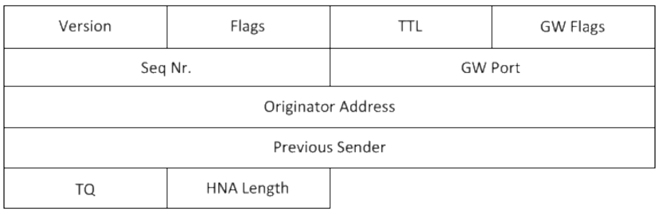


Fig. 1: BATMAN's OGM packet format.

This way all OGM will be received and rebroadcasted by all nodes in the network and all nodes will learn the existence of each other and which nodes are the first hop between them and the other nodes, i.e. the first leg of the path. All nodes and their first hops in their paths are stored in a list called an "Originator List".

When a node which has already received and forwarded an OGM receives the same OGM from another node at a later point - it drops that packet so the network will not get flooded by forwarding the same OGM until its TTL is zero. This is also necessary in order to prevent routing loops.

V. Requirements

Ad hoc networks have some desired characteristics such as quick and inexpensive setup and being independent of communication infrastructure, but they also introduce great challenges regarding security.

A. Scenario

The design and implementation presented in this paper is mostly based on an emergency situation scenario, in which communication infrastructure is unavailable. If there is a major emergency situation such as an earthquake or tsunami, it is likely that parts or the entire communication infrastructure at the scene is destroyed or temporarily down. The remaining communication lines will then probably be congested, such that little communication actually goes through.

In this situation, it is of great importance that Emergency Personnel, such as Paramedics, Firemen, Policemen and the Military, are able to communicate efficiently and therefore independently of the public communication infrastructure. They need this network in order to manage the the operation, and therefore availability is probably the most important trait of this network. Secondly, they should be able to trust the communication on the network – i.e., messages sent are from whom they claim they to be.

Also, being able to authorize new actors on the scene, such as Red Cross, can be critical to the operation. These new actors will probably not have the necessary authentication tokens, i.e. certificates, required by the authentication scheme in the network.

B. List of Requirements

Based on the scenario above these requirements can be extracted and made into general requirements that needs to be addressed by the system design. The work presented here is based on several sources, most prevalent being the research from the OASIS project [2] [10] [1] and Winjum et al. [11].

R1	A node must be authorized in order to get full rights in a network [12], [13]
R2	A node without a recognized authentication token should be able to become authorized if necessary
R3	Networks need a master node which handles access control
R4	Access control (after initial authentication) should not rely on centralized nodes
R5	Different networks should be able to collaborate [11]
R6	Only master nodes can decide access policies of users/nodes
R7	Nodes must not be able to alter access policies they are ruled by

An early study produced security requirements of ad hoc networks demanding that the routing logic must not be spoofed or altered to produce different behavior [12]. This means authorization is required (R1) before someone can partake in routing logic. The OASIS project [2] specifically considered a situation where e.g. NGOs contribute to a rescue operation, which means they need to somehow acquire credentials (R2), but this must be administered by some authority (R3). R4 highlights the need for authenticated nodes to function autonomously. A desire for seamless radio coverage over the area gives us R5. R6 comes from the fact that it is not possble to determine access policies prior to network setup, and R7 states the rather obvious, in that nodes that could alter the access policy would violate R6.

VI. Security Solution Overview

The system design requires nodes to be authenticated and trusted before being allowed into the network. Each node also has to verify their identity periodically, or they are dropped from the network.

The network setup starts with an out-of-band authentication where a master node, hereafter referred to as a Service Proxy (SP), verifies new nodes. How this is done can be up to the application, but let us assume that the actors carrying their communication devices, hereafter nodes, physically meets the SP at the scene and exchange their public key fingerprints.

When a new node is discovered by the SP using regular routing announcements as part of the pro-active routing protocol, the SP will invite the new node to a handshake to establish a trust between the two nodes. The new node will receive the SP's certificate, and will after verifying the fingerprint request a proxy certificate for itself. After verifying the node's fingerprint, the SP will issue a proxy certificate with (possibly) the rights to participate in building the MANET by broadcasting its own and re-broadcasting other trusted nodes' routing announcements.

A. Why use Proxy Certificates?

. The Proxy Certificate (PC) is used to delegate rights on behalf of the issuer. That means that the issuer, i.e. the SP, can choose to delegate all or a subset of its rights to the receiver of the Proxy Certificate. This can be very useful in a situation where the nodes themselves are unable to properly authenticate themselves with their pre-existing conventional X.509 certificate if the SP on the scene has no way to verify their certificates. This can be true if their certificates are issued by an unknown root certificate (CA) or simply if there is no Internet access and the certificate is signed by an unknown entity (unknown to the SP), even if it knows and trusts the root CA.

Also, the SP could be interested in giving the node rights the node would not usually have on this specific scene, depending on the situation. This is easier to achieve when the SP can delegate its own rights.

An important feature of the PC is that the SP can delegate different kind of rights, as long as it is a subset of its own rights, to different nodes. There are countless of different rights that can be useful, given the situation they are used in, but here is a few possible rights/privileges to give the reader an understanding of the possibilities they give:

- Announce itself - let the MANET know of your existence
- Re-broadcast other nodes announcements - reshape the network topology
- Announce a gateway - give the MANET access to another network
- Use the gateway - allow you to communicate outside the MANET
- Send and receive messages with a defined application - full application rights
- Only receive messages from a defined application - limited application rights

If you are setting up a MANET on the scene of a disaster to assist emergency personnel, you could have some actors be able to organize the effort by sending orders/commands to the other actors, while some actors only are allowed to receive the orders. In this situation it might be of great importance to know that only verified nodes are able to give commands, but the importance of getting this information available outweighs the need to verify the nodes/actors receiving this information.

B. Post-Authentication Operation

After being issued with a Proxy Certificate (PC) the newly authenticated node will periodically "broadcast" - unicast to each neighbor - a message containing an ephemeral key and corresponding Initialization Vector (IV), a pseudo-randomly generated nonce, and a digital signature over this message. The ephemeral key is encrypted with the neighbor's public key (hence multiple unicasts instead of an actual broadcast), but the digital signature is generated based on the unencrypted key and the other contents of the message, and is thus identical for all neighbors.

After sending this signed "broadcast" to each neighbor, the node and its neighbors will generate a keystream from the ephemeral key, IV, and nonce. The node will then append two new bytes from this keystream to each routing announcement, and re-broadcasts of neighbors' announcements, sent from this point forward with a sequence number for the recipient to be able to match this "extract" with the keystream at an offset given by the sequence number. The two bytes will then in effect be a one-time password similar to that used by some online banking applications. If this one-time password value is absent or incorrect, the announcement will be dropped and regarded as a spoofing message.

Whenever a routing announcement is re-broadcasted by another trusted node, that node will first replace the sequence number and one-time password that it has verified with the next two bytes of its own key stream. This means that every node only checks its direct neighbor for authentication, which is a design choice. This proposal assumes that because every node is verified by the SP in the first place, all nodes in the network will be able to trust each other, which also means they will trust their neighbors to properly verify their neighbors again.

In order for trusted nodes to learn of newly trusted nodes existence, the SP regularly broadcasts lists containing the id, address and public key of each trusted node in the network. This needs to be done, because before learning about a new node the other trusted nodes will not accept any messages from this node. This means the new node will not be able to exchange its own PC with other nodes directly - only through the SP.

The list, hereafter Authentication List (AL), also adds some web-of-trust like capabilities. The list is signed by the SP, which means the integrity of the list is guaranteed by the SP. This means that if the SP should go offline, e.g. it could be out of range, other trusted nodes in the MANET can continue to broadcast the AL on behalf of the SP - to ensure all nodes in the network know each other. This can be especially important when the network grows large and become fully or partially separated and nodes in one part may not have learnt of the existence of newly trusted nodes yet. It also applies to trusted nodes who have been offline while new nodes have been verified, then re-enter the network while the SP is offline.

VII. Simulations

We have implemented both standard BATMAN and the version with our security enhancements in the network simulation package ns3.

Figure 2 presents Packet Delivery Ratio (PDR) and packet delay results from the simulations running Secure BATMAN, BATMAN and DSDV with 10 nodes and 10 traffic flows.

As seen from Figure 2a, the PDR values of all three routing protocols all well above 80%. Interestingly, Secure BATMAN's PDR values also stay at approximately the same level as the two other protocols. At pause time zero, which is equivalent to continuous node movement, all three protocols show their best behavior with the highest PDR values. This is probably due to the fact that they all are ad hoc network protocol tailored for networks with high node mobility.

When looking at the end-to-end latency in Figure 2b it is surprisingly the Secure BATMAN protocol which has the best results.

VIII. Prototype

We have implemented our proposed protocol changes by modifying the BATMAN code distributed with a recent Ubuntu Linux distribution.

A. Initialization Phase

Figure 3 presents neighbor discovery results for both the original (Fig. 3a) and modified (Fig 3b) version of BATMAN. The two graphs shows the time in seconds on the y-axis and the trial/run number on the x-axis. The two colored lines on the graphs show the results from first neighbor discovery until the first neighbor is added to routing table (green line - marked with "x") and until both nodes are added to the routing table (red line - marked with "+").

The results from the original protocol, shown in Figure 3a, shows high variance in the time needed to add one and two nodes to the routing table. For 7 out of 10 "first nodes" the time needed is relatively equal, being about one second. For both nodes to be added however, there are much more variance - variying from the best possible time, i.e. equal to adding one node, and up above 3 times longer than adding one node. '

Figure 3b shows the results from the modified version proposed in this thesis. These results indicate that the behaviour of the modified version seems to correlate with the behaviour expected from the hypothesis. A seemingly constant of about two seconds seems to be added to the process of adding both nodes to the routing table.

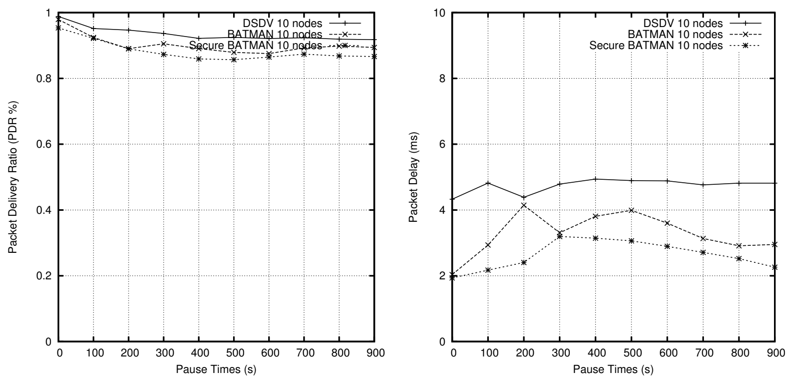
Another interesting observation is that the time variance seems to be much less than that of the original version. This might be because the authentication handshake and the keystream sharing happens in a separate thread from the regular BATMAN operations, meaning the BATMAN protocol continously receives routing announcements to process while the Authentication Module (AM) handles its part. The idea being that while the AM thread runs the BATMAN thread "gets ready" to do its part of the job.

B. Route Convergence

The results of the second test are shown in Figure 3c. In this figure, the axes are the same as in the figures above: y-axis shows the time in seconds, and the x-axis shows the trial run. The red line shows the performance of the original implementation, while the green line shows the modified.

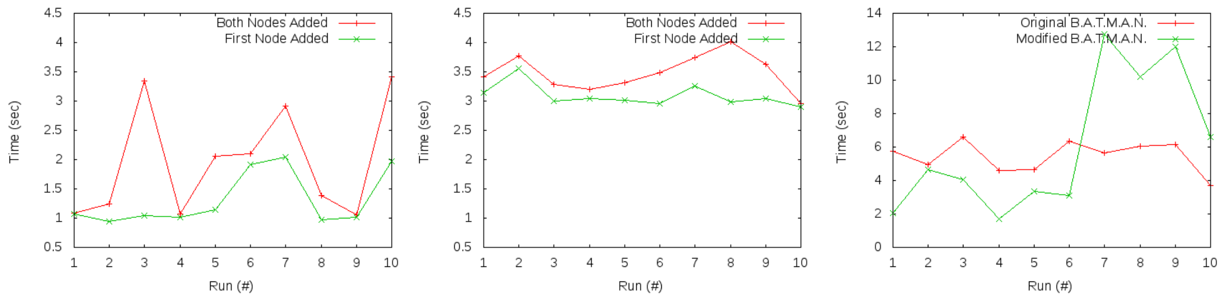
As indicated earlier, this test's results are somewhat unclear. While the results using the original implementation seems relatively uniform, with only about 1 second variance, the results from the modified implementation are highly irregular.

Looking through the logs from this test one thing become apparent. With different hardware on the different nodes in the network, their wireless cards send at different levels of transmission power, meaning that while one node can receive



(a) PDR with varying pause times. (b) Packet delay with varying pause times.

Fig. 2: Simulations results from BATMAN, Secure BATMAN and DSDV (10 nodes and 10 source and sink pairs)



(a) Original BATMAN (b) Modified BATMAN (c) Routing path convergence time

Fig. 3: Neighbor discovery for original and secure BATMAN, and routing path convergence

packets from a “stronger node”, the packets sent might not be received by the other nodes.

The BATMAN protocol messages (routing announcements) are sent quite often, depending on the number of re-broadcasts being sent, meaning the time from when a node is within transmitting range and until its broadcasts are received by nodes within its transmitting range will be quite short. The AM messages however, was mostly tested in an ideal environment where most packets were received, so this was not properly accounted for. Therefore, if a routing announcement from a “stronger node” is received by a “weaker node”, the weaker node might send its keystream material without the other node receiving it.

Re-transmitting mechanisms based on guessing that the receiving node has not received the AM messages are in place, but as the mechanism wait until it beleives the other node has not received, instead of knowing it instantly. This can of course be managed adding ACK’ing to each AM message, which was not added initially because of the wish to minimize overhead. This however, might have to be re-evaluated.

Another thing to notice is how multiple trial runs using the modified version actually performed better than the original version. This is impossible to explain talking about the design and implementations themselves, but is probably most accuratly explained in the terms of external environment.

IX. Discussion

The proposed system design uses a novel solution to continuously verify routing announcements received from one’s neighbors.For this system to be used on typical mobile devices with all their constraints, limitations on computing power, battery lifetime, and saturation in the wireless network must be acknowledged.

Because all nodes in a MANET using a pro-active routing protocol broadcast their routing announcements and forward all received routing announcements, the network traffic will increase exponentially to the amount of nodes in the network and how closely bound they are. Therefore all routing announcements need to be as small as possible. A typical signature is usually one or two orders of magnitude larger than a regular routing announcement, so by adding a signature to the routing announcement - most of the data sent in the network would be signature data. This is far from ideal.

The first solution that one would think of would be to only sign a very few of the announcements, periodically. This however, would be totally disastrous. This would have no protection against spoofing attacks whatsoever, as an attacker could wait for a legitimate node to send a signed announcement and then send his own fake announcements spoofed with the legitimate node's address.

The solution proposed in this paper solves the problem in a different manner. Since each node and its neighbors generate a key stream that can be used to verify messages from that node, only messages with a correct, previously unused, "one-time password" will be accepted and forwarded by any neighbor. Furthermore, since the keystream has to be renewed periodically, any node not possessing the correct proxy certificate will be dropped from the network upon renewal.

This scheme is fully based on trust. You trust that your trusted nodes will only send you its own annoucement (correctly) and rebroadcast only its trusted nodes announcements without modification. If for some reason a trusted node should behave maliciously, this scheme will not detect this and allow the trusted node to potentially disrupt the network.

X. Conclusion

We have presented a security extension to the BATMAN ad hoc routing protocol which handles controlled network admission and prevent unauthorized nodes from influencing routing decisions in the network. Our ns-3 simulations indicate that the security mechanisms do not place an undue burden on the network nodes, and our protoype implementation confirms that although further refinements are desirable, BatCave represents a viable securty solution for ad hoc networks.

References

[1] A. Nyre, M. Jaatun, and I. Tøndel, "A secure MANET routing protocol for first responders," in *Security and Communication Networks (IWSCN), 2009 Proceedings of the 1st International Workshop on*. IEEE, 2009.

[2] I. S. Svagård (editor), "Information security for field workers in crisis situations," SINTEF ICT, http://www.oasis-fp6.org/documents/OASIS_SP24_DDD_253_security_SIN_1_0_pub.pdf, Tech. Rep., 2008.

[3] P. K. Sharma, "Short-Lived Certificates as a Mobile Authentication Method," MSc Thesis, 2009. [Online]. Available: http://orbit.dtu.dk/getResource?recordId=245323&objectId=1&versionId=1

[4] M. Pitkanen and H. Mikkonen, "Initalizing mobile user's identity from federated security infrastructure," in *Proceedings of the Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 08)*, 2008, pp. 390–394. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/UBICOMM.2008.64

[5] M. Reineri, C. Casetti, and C.-F. Chiasserini, "Routing protocols for mesh networks with mobility support," in *Proceedings of the 6th international conference on Symposium on Wireless Communication Systems*, 2009, pp. 71–75. [Online]. Available: http://ieeexplore.ieee.org/iel5/5277434/5285213/05285344.pdf?arnumber=5285344

[6] M. Abolhasan, B. Hagelstein, and J. C.-P. Wang, "Real-world performance of current proactive multi-hop mesh protocols," in *15th Asia-Pacific Conference on Communications (APCC09)*, Oct. 2009, pp. 44–47. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5375690

[7] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist, "X.509 Proxy Certificates for Dynamic Delegation," in *Proceedings of the 3rd Annual PKI R&D Workshop, Gaithersburg MD, USA*, 2004.

[8] A. Neumann, C. Aichele, M. Lindner, and S. Wunderlich, "Better Approach To Ad-Hoc Networking (B.A.T.M.A.N) draft-wunderlich-open-mesh-manet-routing-00," *Network Working Group*, Last accessed December 19, 2010, http://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00.

[9] O. Mesh, "*Why starting B.A.T.M.A.N.?*" *open-mesh.org*, Last accessed december 19, 2010, http://www.open-mesh.org/wiki/why-starting-batman.

[10] I. Tøndel, M. Jaatun, and A. Nyre, "Security requirements for MANETs used in emergency and rescue operations," in *Security and Communication Networks (IWSCN), 2009 Proceedings of the 1st International Workshop on*. IEEE, 2009.

[11] E. Winjum, P. Spilling, and Ø. Kure, "Ad Hoc networks used in emergency networks : the Trust Metric Routing approach," FFI Rapport, Tech. Rep., 2006.

[12] B. Dahill, B. Levine, E. Royer, and C. Shields, "A secure routing protocol for ad hoc networks," *Electrical Engineering and Computer Science, University of Michigan, Tech. Rep. UM-CS-2001-037*, 2001.

[13] K. Sanzgiri, B. Dahill, B. Levine, C. Shields, and E. Belding-Royer, "A secure routing protocol for ad hoc networks," *Network Protocols, IEEE International Conference on*, vol. 0, p. 78, 2002.