# End-to-End Security Mechanisms for the Optimized Link State Routing Protocol for Wireless Ad Hoc Networks

A dissertation submitted to the Department of Computer Science,
Faculty of Science at the University of Cape Town
in fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

— Stephen Warren Asherson —

Supervisor
Adjunct Professor Andrew Hutchison

UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

January, 2008

# ABSTRACT

Routing protocols designed for wireless ad hoc networks are, in general, highly vulnerable to various forms of security attacks. A routing protocol is vital to the functioning of a wireless ad hoc network, and hence, security needs to be present to negate any potential malicious influences. However, providing efficient security mechanisms for such routing protocols is still viewed as being a considerable challenge.

In this dissertation, the focus lies on the Optimized Link State Routing (OLSR) protocol, a proactive protocol which relies heavily on broadcast transmissions. This study investigates the use and feasibility of several end-to-end security mechanisms for the OLSR protocol, with specific interest in the overhead and performance penalties incurred by such security mechanisms. In general, the main focus of the security mechanisms fall on authentication, integrity, and replay protection for the OLSR message. More specifically, we investigate the use of a one-time signature scheme known as Hash to Obtain Random Subset (HORS), as well as an extended version of HORS, as a broadcast digital signature for OLSR messages.

For the experimental purposes of this study, an existing OLSR implementation was used as a basis for implementing a *security-aware* version of OLSR, incorporating our chosen security mechanisms. The experiments were performed on a 9 node indoor wireless mesh testbed, and consisted of testing both the standard OLSR protocol, as well as our security-aware version of the protocol. For each of the protocol versions tested, various performance aspects were recorded, allowing comparisons amongst the different versions to take place.

The results obtained from the experiments indicate that the chosen security mechanisms add a significant amount of overhead, particularly in the department of traffic overhead. Whilst our extended version of the HORS signature scheme performed better than that of the standard HORS scheme, it was found that the performance of both schemes degraded rapidly as the hop-count, between any two given communicating entities, tended to increase. This performance degradation exists primarily due to the key management problems of the HORS one-time signature scheme, particularly regarding the process of public key distribution in a wireless network environment which lacks reliable broadcast transmissions and is subject to packet loss. The results from the experiments provide insight into the use of various security mechanisms, and also demonstrate the engineering challenges associated with securing a routing protocol based on IEEE 802.11 broadcast transmissions.

# ACKNOWLEDGMENTS

*First and foremost to my supervisor, Andrew Hutchison, for his ongoing guidance, support, and endless patience throughout the duration of this dissertation*

*To Mom, Dad, Mandy, Diana, Craig, and my girlfriend, Brice Salence, for their love and unconditional support which has made this process so much easier*

*To the Data Networks Architecture (DNA) laboratory, and more specifically to the founder Professor Pieter Kritzinger, for all the support and opportunities afforded to me*

*To my fellow lab member, Andrew Symington, for his friendship, and his continuous and exceptional help throughout this project*

*To all my lab friends, for all the good times and memories, and for making these past 2 years so much more than just academia*

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ALGORITHMS

CHAPTER 1

# INTRODUCTION

In recent times, wireless networking has witnessed a phenomenal growth in adoption. Wireless networking has allowed communications to move from the fixed regime of wired networks, to a regime which promotes mobility and spontaneity. By employing the wireless medium, communications can exist between parties in areas previously deemed infeasible; one can see how this characteristic of wireless networking can be beneficial in emerging communities, with particular reference to providing services in rural areas - areas often lacking in infrastructure.

Wireless Fidelity (Wi-Fi) - the industry name used to refer to the Wireless Local Area Network (WLAN) technology - is in widespread use today. In its most typical form, Wi-Fi connects wireless clients to centralised wireless access points, located within the same building or room. Albeit wireless networking, a centralised networking model can place restrictions on communications in the absence of reachable wireless access points. In light of this restriction, the standards governing the Wi-Fi technology - the IEEE 802.11 group of standards [33] - also specify a decentralised networking model, permitting communications solely between Wi-Fi clients, commonly known as a wireless ad hoc network [16, 10]. Not only is communication possible between directly connected wireless clients, but also, possibly between nodes situated out of wireless range of one another. This form of communication, between out of range nodes, is possible via a multi-hop communication channel created by intermediate nodes, if such intermediate nodes exists. Figure 1.1 shows a simple ad hoc network of 3 wireless nodes, intermediate node B forms a multi-hop communication channel for nodes A and C which lie out of wireless range of each other.

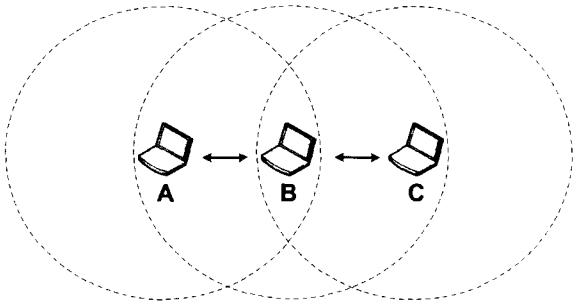



Figure 1.1: Wireless Ad Hoc Network of 3 wireless Nodes.

Whilst the decentralised networking model afforded by ad hoc networks permits spontaneous networking, in the absence of a fixed infrastructure, the role of enforcing security amongst such communications is left entirely to the client nodes themselves. Not only are security threats prominent amongst communications between directly connected parties, these threats are further magnified due to the multi-hop communication nature of wireless ad hoc networks. In multi-hop communications between parties, intermediate nodes, along the multi-hop communication channel, are in prime position to exploit any present weakness by executing security attacks on the communications. Thus, security needs to be present to ensure that end-to-end communicating parties are protected from potential security attacks, facing the communications between them.

## 1.1 RESEARCH OUTLINE

Whilst this research is concerned with a very specific topic, it forms part of several broader research fields. The research performed as part of this dissertation requires a general understanding of these fields. Figure 1.2 outlines the research fields corresponding to this work, with the eventual emphasis falling on the specific research field which forms the topic of this dissertation.

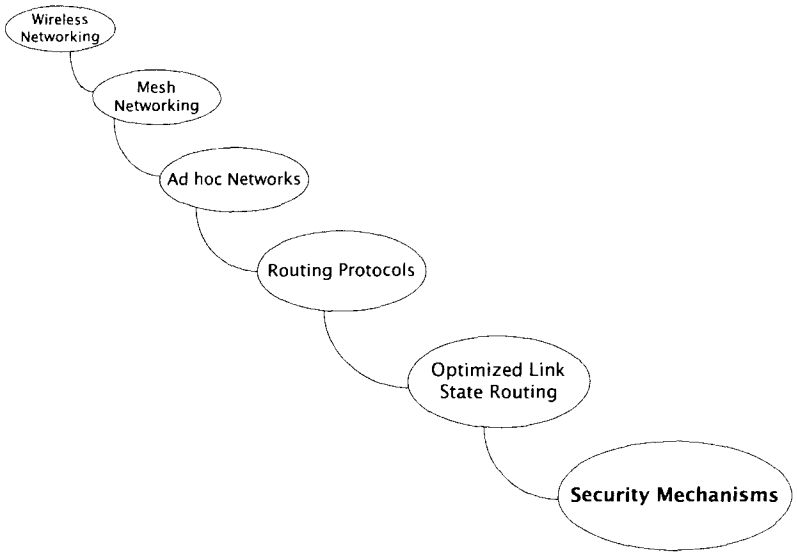



Figure 1.2: Dissertation Research Outline.

### 1.1.1 Routing Protocols for Wireless Ad Hoc Networks

Owing to an absence of any infrastructure, and the limited wireless range of a given client in an ad hoc network, it may be the case where data exchange, between any 2 given nodes, is only possible along a multiple hop route via intermediate wireless clients, acting as routers. Routing protocols allow wireless clients to establish multi-hop routes to other available destinations in the network. Each wireless clients in the network is responsible for routing and forwarding data packets.

### 1.1.2 Security of Wireless Ad Hoc Routing Protocols

The process of routing in a wireless ad hoc network forms the backbone for data communications between the hosts in the network. Hence, the correct operation of an ad hoc routing protocol is of considerable importance. Several constraints exist in the face of wireless ad hoc networks [10]: wireless clients are limited by power constraints, particularly when small devices (PDAs, cellular phones) are involved; bandwidth constraints are also present, considering the shared, relatively slow - when compared to wired networks - wireless medium used. Wireless ad hoc routing protocols are therefore often designed with efficiency as a priority, leaving the security of such protocols an afterthought.

Routing protocols designed for ad hoc networks are vulnerable to many security threats [5, 9], compromising the proper operation of such protocols. These threats make the securing of such protocols very important. In fact, Hoeper et al. [29] list the process of *securing ad hoc routing protocols*, as one of the four main security goals facing ad hoc networks. The remaining three goals include: 1) authentication amongst communicating devices; 2) the secure establishment of security session keys; and 3) the secure storage of security key data on network devices.

## 1.2 RESEARCH QUESTIONS

Wireless ad hoc routing protocols often require data to be exchanged between hosts several hops apart. To provide an end-to-end security association between hosts exchanging routing data, additional security data corresponding to the routing data must also be exchanged. In the context of the Optimized Link State Routing (OLSR) [11] protocol, a routing protocol relying on regular broadcasts of routing control messages, additional security data for each control message can amount to a significant increase in overhead. This increase in overhead comes in terms of routing traffic overhead, as well as the additional processing involved, of the security data, at each hop.

Efficient broadcast authentication, in the context of ad hoc networks, is still an open research problem [46, 51]. The research area of one-time signatures [45, 42] aims to provide efficient, asymmetric signature primitives. Whilst one-time signatures can provide efficient digital signature operations, these schemes are often hindered by several factors. Some of these limitations include: large key and signature sizes; limited number of signatures per key pair; and general key management problems. In [46], Naor et al. ask the question of whether one-time signature schemes have become practical, in the face of such limiting factors.

In the context of end-to-end security in wireless ad hoc routing protocols, two research questions are evident: 1) is it practical to enforce end-to-end security mechanisms in a wireless ad hoc routing protocol? And 2) is the application of one-time signature schemes, in the broadcast environment of an ad hoc routing protocol, practical in providing asymmetric signatures of routing control data?

## 1.3 APPROACH

### 1.3.1 Questions Addressed

The importance of security mechanisms for wireless ad hoc routing protocols has been touched upon, briefly. In this work, we investigate end-to-end security mechanisms for the Optimized Link State Routing (OLSR) protocol, and determine the practicality and overhead of such mechanisms. In particular, we investigate the practicality of a one-time signature scheme, known as Hash to Obtain Random Subset (HORS) [53], as a broadcast authentication mechanism for the OLSR routing protocol.

### 1.3.2 Research Methods

In order to address the questions posed above, a method of testing is required to allow one to draw conclusions on the use of end-to-end security mechanisms in the OLSR protocol. For this dissertation, experiments have been undertaken in an attempt to obtain measured results. The experiments were performed on a live wireless mesh testbed. From the results, comparisons are drawn between the use of standard OLSR, and that of a secured version of OLSR.

## 1.4 OUTLINE OF THE DISSERTATION

In chapter 2, we discuss background literature to the fields of wireless networking, wireless ad hoc routing protocols, and the security of such routing protocols. In chapter 2, we also touch on security primitives relevant to the work that has been done in this dissertation. In chapter 3, we introduce a security scheme for OLSR, incorporating several end-to-end security mechanisms, including two different signatures schemes, one of which is an extended version of the HORS one-time signature scheme. Chapter 4 discusses the implementation details of: the security primitives and mechanisms required in this work, our OLSR security scheme, and how the security mechanisms have been incorporated into OLSR. In chapter 5, we disclose the details of our wireless hardware testbed, used for experimental purposes in this work. In chapter 6, we discuss the details of each test that formed part of the experiments, as well as specify values for various parameters used in the tests. Chapter 7 reveals and discusses the results from the experiments. In chapter 8, conclusions are drawn and recommendations for future works touched upon.

CHAPTER 2

# BACKGROUND AND RELATED WORK

## 2.1 INTRODUCTION

## 2.2 WIRELESS NETWORKS

Broadly speaking, a wireless network [21] can be defined as a number of devices communicating using Radio Frequency (RF) waves. Wireless technology has existed for quite some time. In fact, modern wireless communication dates back to 1896 with a demonstration of wireless telegraphy by Marconi [40]. Recently, an increase in demand for mobile communications coupled with a decrease in cost of wireless commodities has seen a boom in the application of the wireless technology as a network medium. Wireless equipment vendors have been quick to realise the strong potential of wireless networks in both a commercial and residential sense. Not only has the convenience factor of the wireless technology accelerated its adoption, the cost benefits provided by a wireless network over a typical wired infrastructure have not gone unnoticed. To provide interoperability amongst wireless devices from different vendors, standards that govern the technology are essential. The Institute of Electrical and Electronics Engineers (IEEE) [60] has been instrumental in advancing, and defining standards for wireless technologies.

A wireless network may exist in different architectures and sizes. The characteristics of a wireless network will typically be governed by the underlying wireless technology and the network's application. The following network architectures are the most commonly found wireless networks using IEEE 802.11 wireless technologies:

- Wireless Personal Area Network (WPAN) : A WPAN is a network formed by devices within a very close proximity of one another. One example is a pair of cellular phones communicating over Bluetooth® [59], a short-range wireless technology. The WPAN technology is governed by the IEEE 802.15 working group [35].
- Wireless Local Area Network (WLAN) : A WLAN is the most common form of wireless network in use today. In its most conventional form, a WLAN consists of an access point providing wireless access to wireless clients within the same room or building. The IEEE 802.11 working group [33] governs the WLAN technology.

- Wireless Metropolitan Area Network (WMAN) : A WMAN specifies a network operating over longer distances than that provided by a WPAN or WLAN. WMAN distances can range from several building blocks to entire cities. The WMAN technology, commonly known as WiMAX (Worldwide Interoperability for Microwave Access), is defined by the 802.16 family of standards, controlled by the IEEE 802.16 working group [36]. A potential application of the 802.16 technology is in providing last-mile broadband connectivity to consumers.

### 2.2.1 WLAN - The IEEE 802.11 Standard

The IEEE 802.11 family of standards define Medium Access Control (MAC) and Physical layer (PHY) specifications for Wireless Local Area Networks. Whilst the underlying technology of a WLAN (802.11) is significantly different to that of a wired LAN, IEEE 802.11 is designed such that, to higher layers in the stack, it appears as a regular IEEE 802 LAN [32]. The family of 802.11 standards incorporates various amendments and extensions to the original 802.11 wireless standard, the table below gives an overview of the most widely used and accepted of these standards, compiled from [15, 21].

| Technology | Max. Data Rate | RF Band | Summary |
|---|---|---|---|
| 802.11 | 2 Mbps | 2.4 GHz | Slow data rates, legacy standard. |
| 802.11a | 54 Mbps | 5 GHz | High data rates and 8 available channels. Provides better protection against interference. Standard ratified in 1999. |
| 802.11b | 11 Mbps | 2.4 GHz | Three available channels. Standard ratified in 1999. |
| 802.11g | 54 Mbps | 2.4 GHz | Three available channels. Backward compatible with 802.11b. Standard ratified in 2003. |
| 802.11n | 300 Mbps | 2.4 or 5 GHz | Draft v2.0 - yet to be ratified as a standard. |

The 802.11b and 802.11g standards have become very popular due to the lower cost of these products when compared to 802.11a. The 802.11b/g standards however are more prone to interference as they operate in the license free 2.4 GHz ISM (Industrial Scientific Medical) band, this band also being common to various other commonly used RF devices. 802.11n [4] is an emerging technology which aims to offer longer operating ranges and higher throughputs via a multiple transmitter and receiver system known as MIMO (Multiple-Input Multiple-Output). At the time of this writing, the latest draft specification for the 802.11n technology is version 2.0.

### *Components of the IEEE 802.11 Architecture*

#### The Basic Service Set

A Basic Service Set (BSS) is the core component of the IEEE 802.11 architecture. A BSS can be defined as a coverage area within which member stations can remain in communication. figure 2.2 shows two separate BSSs. In most cases, a basic service set is comprised of a wireless Access Point (AP) which provides connectivity and services to wireless stations that lie within wireless range of the AP.

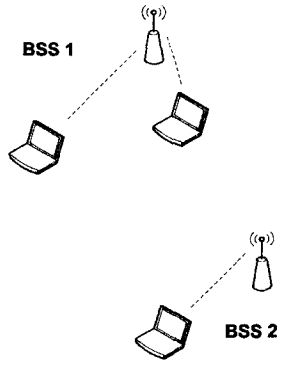



Figure 2.1: IEEE 802.11 Basic Service Sets.

#### The Independent Basic Service Set

An Independent Basic Service Set (IBSS) is the most basic component of the IEEE 802.11 architecture. An IBSS can be defined as a BSS without an infrastructure/AP. An IBSS consists of wireless stations which can communicate directly with one another.

#### The Extended Service Set

Whilst a BSS may provide connectivity amongst its wireless stations, it could be the case that connectivity is required between wireless stations that lie in different BSSs. 802.11 uses the concept of an Extended Service Set (ESS) to connect separate BSSs in forming an arbitrary sized wireless network. The BSSs are connected via a backbone network, known as the Distribution System (DS); 802.11 does not specify a specific medium for the DS. APs acts as layer 2 bridges, whereby 802.11 logically separates the wireless medium from the Distribution System medium.

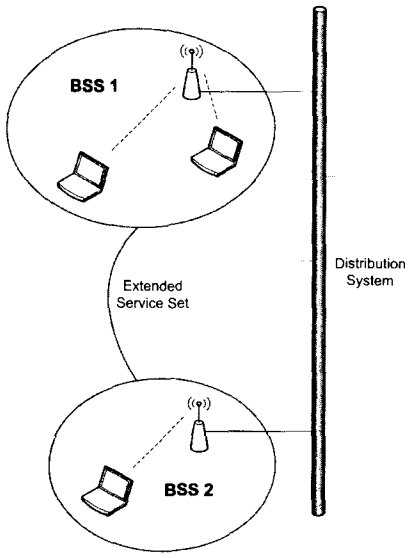



Figure 2.2: IEEE 802.11 Extended Service Set.

### 2.2.2 IEEE 802.11 Security Considerations

Since the standardisation of the IEEE 802.11 standard, a major obstacle standing in the way of widespread adoption has been the security vulnerabilities of the open wireless medium. For instance, consider communications between two wireless entities, the following are some security vulnerabilities that exist in the absence of security:

- Forging: a wireless client can forge data in an attempt to parade as someone else.
- Tampering: traffic being sent over the wireless medium can be manipulated in a negative manner.
- Eavesdropping: potentially sensitive information can be heard by others listening over the open wireless channel.

To counter these threats, security mechanisms were incorporated into the initial IEEE 802.11 standard. This section aims to provide a brief overview of the security mechanisms present in the IEEE 802.11 and IEEE 802.11i [34] standards. Much of the overview provided in the section has been compiled from work done in [49, 32, 34, 28, 21]. The original IEEE 802.11 standard provides the optional use of Wired Equivalent Privacy (WEP).

***Wired Equivalent Privacy (WEP)***

The WEP protocol provides three main security mechanisms to the operation of 802.11: authentication; integrity; and confidentiality.

**Authentication**

Two authentication options exist in the original 802.11 standard. The first option is known as Open System authentication. This form of authentication is the most simple form of authentication and does not provide any form of security. Open System authentication is a two step protocol where it essentially consists of an authentication request and reply between a requesting entity and an authenticator, typically an AP. The identity of the requesting station in the request frame is the only form of authentication material used in the exchange.

The second authentication option is Shared Key authentication. This option requires that an encryption key be pre-shared on the parties involved in the authentication process. Shared Key authentication is essentially a four step protocol between a requesting station (STA) and an authenticator (AUTH):

1. STA to AUTH: Request to authenticate.
2. AUTH to STA: Random challenge.
3. STA to AUTH: Challenge response (challenge encrypted with shared key).
4. AUTH to STA: Result of authentication.

**Confidentiality**

The initial WEP standard made use of a 40-bit encryption key to encrypt wireless traffic. Vendors were quick to improve on this by offering 104-bit key solutions. WEP uses the shared key appended with a random 24-bit Initialisation Vector (IV) to encrypt 802.11 data frames. The IV is sent in cleartext between the communicating parties to allow the recipient to append it to the shared key for encryption/decryption purposes.

**Integrity**

WEP uses a 4-byte Integrity Check Value (ICV). The ICV is a Cyclic Redundancy Check (CRC) and is computed over the bytes in the 802.11 data frame. As added security, the ICV value is also encrypted using the pre-shared encryption key.

**Security Flaws of WEP**

The WEP protocol of the original 802.11 standard did not turn out to be a success. Many security flaws of the protocol surfaced and a lack of confidence in its use left it obsolete. A detailed overview of the flaws discovered in WEP is given in [49]. Realising this problem, the IEEE commissioned a task group (task group i) dedicated to providing improved security for the 802.11 wireless technology. The task group led to the ratification of the IEEE 802.11i security amendment in June 2004.

### *The IEEE 802.11i Security Amendment*

The IEEE 802.11i standard is an extension to the 802.11 standard and specifies additional security mechanisms provided at the Medium Access Control (MAC) layer. The 802.11i extensions allow for the establishment of what is called a Robust Security Network Association (RSNA).

## 2.2.3 RSNA Algorithms

The IEEE 802.11i amendment introduces what is known as RSNA algorithms which provide security enhancements over the original 802.11 standard in the areas of station authentication, authorization, key management, and data confidentiality.

### *Station Authentication*

IEEE 802.11 RSNA algorithms support two methods for stations to perform authentication: 802.1X port based authentication using an upper-layer EAP [8] authentication method; and Pre-Shared Key (PSK) authentication. IEEE 802.1X is a port based network access control protocol and uses the uncontrolled/controlled port model to authorize and control data flow between an authenticator/Access-Point (AP) and any supplicant(s). If 802.1X EAP authentication is used, the 802.1X controlled port is blocked and restricts general data flow between the supplicant and AP until a higher-layer EAP authentication procedure completes successfully. The EAP authentication procedure (usually EAP-TLS) makes use of an Authentication Server (AS), in most cases a RADIUS [54] server. A single device can act as both an AP and AS, however, if this is not the case, it is assumed a secure connection exists between the AP and AS. The entire authentication process can be broken into several stages (see figure 2.3), a brief 3-stage overview is given below:

1. A discovery and association phase between a supplicant and an AP. This initial authentication is Open Systems authentication and is kept for backward compatibility. The association between the station and AP is used to establish security parameters prior to further authentication. At this stage the supplicant is weakly authenticated and associated, however the 802.1X controlled port remains blocked for general data packets.

2. If 802.1X authentication is used, the EAP authentication process commences. The supplicant and AS then perform mutual authentication using a higher-layer EAP authentication protocol (most commonly EAP-TLS), whilst the AP serves as a relay point. During this stage the supplicant and AS generate and establish a common Pairwise Master Key (PMK); in addition, the AS provides the AP with the required material allowing the AP to generate the same PMK. If static pre-shared key authentication is used, the pre-shared keys can serve as the PMK and the 802.1X EAP stage can be skipped.

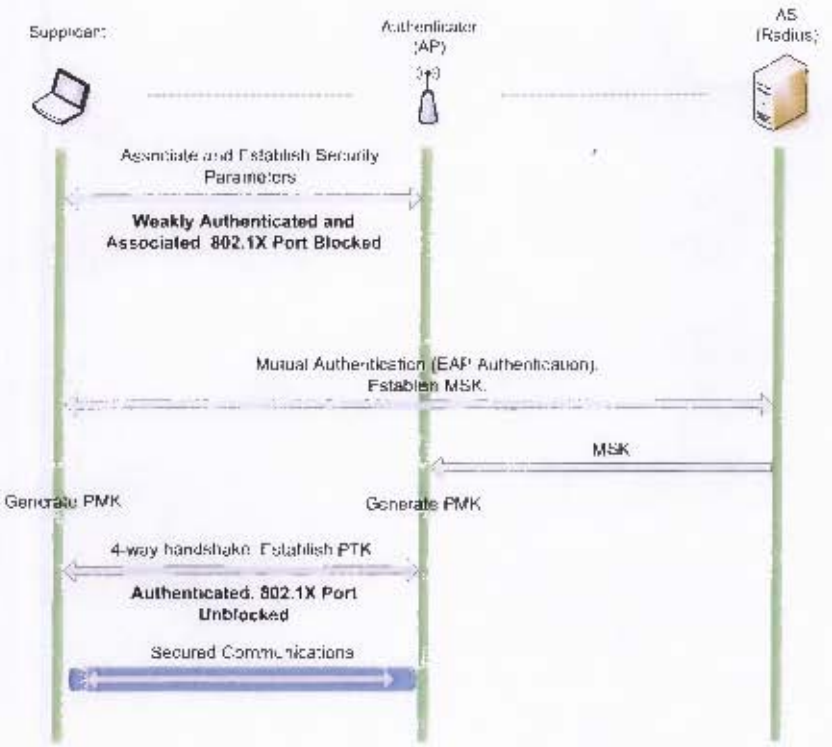



Figure 2.3: IEEE 802.1x EAP Authentication.

3. The final stage of the RSNA establishment procedure consists of a 4-way handshake between the supplicant and AP. The supplicant and AP first confirm the existence of a common PMK. A fresh Pairwise Transient Key (PTK) is then established and cipher suite parameters selected for the following data session. This stage may also be used to distribute a Group Transient Key (GTK) to the supplicant. Upon successful completion of this stage, the AP and supplicant have established a security association between them, and the 802.1X controlled port is unblocked to permit general data traffic.

### *Data Confidentiality and Integrity*

The IEEE 802.11i amendment supports two algorithms for data privacy:

1. The Temporal Key Integrity Protocol (TKIP) offers many enhancements over the Wired Equivalent Privacy (WEP). In summary, TKIP uses a 128-bit temporal key which when combined with the MAC address of a station and a 16-octet IV, is used for encryption. Although TKIP also uses the RC4 stream cipher for encryption, like WEP, one significant difference is that TKIP periodically refreshes temporal keys. TKIP uses an 8-bytes Message Integrity Code (MIC) as an improvement over WEP's ICV in order to ensure the integrity of an 802.11 data frame. TKIP is only a temporary solution to improve on the shortcomings of WEP.

2. Advanced Encryption Standard (AES) [18] in Counter Mode CBC-MAC Protocol (CCMP) mode. This method offers the highest security in terms of encryption and integrity. A 128-bit AES key is used for encryption; the AES-CCMP algorithm also provides a MIC over the 802.11 data frame for integrity purposes.

### 2.2.4 Wireless Network Architectures

Three common wireless architectures (WPAN, WLAN, WMAN) have already been discussed briefly, these types of architectures are classified according to the size and range of the network. The remainder of this section describes how the architecture of a wireless network can be further classified according to the structure and layout of the communicating entities forming the network. In this section, two architectures are explained: Infrastructure Wireless Networks; and Wireless Mesh Networks.

#### *Infrastructure Wireless Networks*

An infrastructure wireless network is formed by users connecting to a central infrastructure point using a wireless technology. This form of wireless network is centralized as clients receive network connectivity from a central node, known as an access point; see figure 2.4. The access point maintains a backhaul connection (in most cases a wired connection) to another network, or possibly the internet.

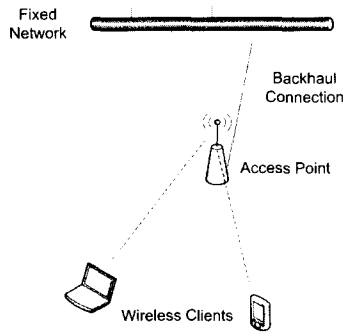



Figure 2.4: A generic infrastructure wireless network.

The wireless clients communicate directly with the access point, thus the access point can only serve clients that are in direct wireless range. This type of wireless network is the most conventional, and most commonly found wireless network in use today.

### *Wireless Mesh Networks*

Infrastructure wireless networks require wireless clients to perform all communications via an access point, this imposes two restrictions on the network: clients located near one another do not communicate directly with each other, but rather via the AP; and clients located out of range of the access point have no wireless connectivity.

A Wireless mesh network (WMN) is a type of wireless network which aims to provide a solution to the restrictions mentioned above. WMNs are multi-hop networks, and are generally more decentralized than infrastructure wireless networks. WMNs can be classified into three main architectures: Infrastructure Wireless Mesh Network, Client Wireless Mesh Network, and Hybrid Wireless Mesh Network [2].

An Infrastructure WMN is formed by a central routing infrastructure consisting of dedicated wireless mesh routers. The mesh routers form a multi-hop network which extends communications to wireless clients. One or more mesh routers act as gateways to a backhaul network or the internet. Mesh routers communicate with one another using a wireless technology such as 802.11 or 802.16. This WMN architecture is depicted in figure 2.5

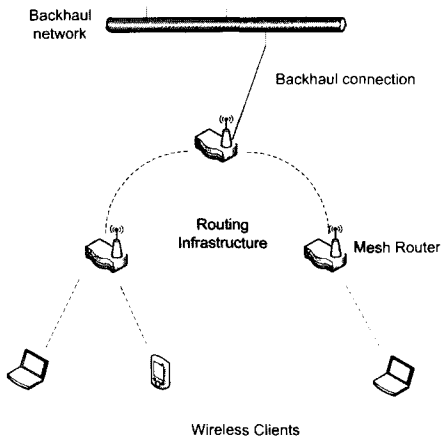



Figure 2.5: An Infrastructure wireless mesh network.

The mesh routers serve two purposes: 1) to act as wireless access points to the clients; and 2) to route traffic through the routing mesh infrastructure to gateway nodes. This type of WMN architecture provides many benefits over a standard access point wireless network. From a cost point of view, each access point no longer requires a wired backhaul connection. Instead, the backhaul can be reached via multiple wireless hops. Wireless connectivity also allows access points to be installed in areas that were previously inaccessible, hence, allowing extended connectivity. A promising application of these networks is the provision of last mile access to wireless broadband services.

The second type of mesh network mentioned, the client WMN, differs quite substantially to that of an infrastructure WMN. In a client WMN, no static infrastructure is present. The client nodes collectively form a decentralized, multi-hop network. Nodes are free to roam, joining and leaving the network dynamically as a result. The nodes cooperate by performing network

services such as data forwarding and routing; this permits communication between devices out of direct wireless transmission. This type of network architecture is commonly known as an ad-hoc network [16], and is depicted in figure 2.6.

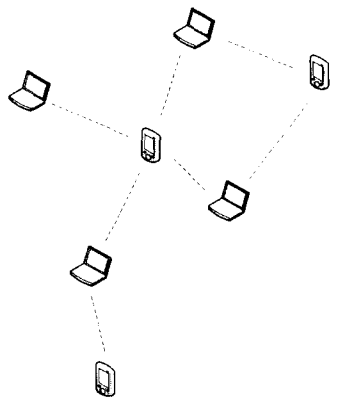


Figure 2.6: A client wireless mesh network.

This type of architecture is used when a small, spontaneous network is required to provide communications between parties where no infrastructure may be present. This type of collaboration networking is also referred to as spontaneous networking [17]. Several applications of an ad hoc network architecture are listed below:

1. Military Operations: Ad hoc networks were primarily used military environments [10, 2]. In a military scenario there are many situations where the presence of a communications infrastructure is not possible. Ad hoc networks are ideal in these circumstances as they allow spontaneous communications between military personal.

2. Emergency Situations: In an emergency, spontaneous communications between aiding parties is often required to be established rapidly [72, 71]. An ad hoc network exerts this feature, providing a number of parties with a means to communicate almost instantly.

### *Hybrid Wireless Mesh Network*

A hybrid WMN is a mix between an infrastructure WMN and a client WMN. In this type of architecture, an infrastructure of mesh routers still serve as the main communications backbone of the network; however, communications can be further extended by the client nodes as they can also act as mesh routers. This type of architecture provides both the stability of an infrastructure network, as well as the dynamic nature of a client mesh network. This architecture is shown in figure 2.7.

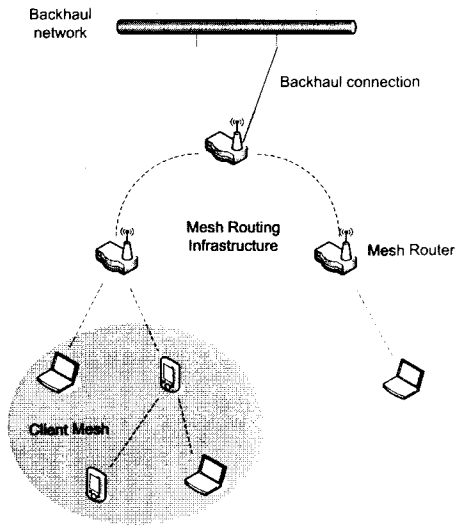



Figure 2.7: A hybrid wireless mesh network.

## 2.3 WIRELESS AD HOC ROUTING PROTOCOLS

In a network, when two devices communicate with one another, it may be the case where the devices are not directly connected. Hence, the communications between the devices may traverse one or more intermediate devices, known as routers. The routers are responsible for forwarding data to the next device until the data reaches the intended destination(s). When forwarding data, routing is the process of determining the next intermediate device which lies on route to the intended destination(s).

Owing to the dynamic nature of an ad hoc network and the lack of infrastructure present, the task of routing data between source and destination is less trivial than in a regular static network. Instead of dedicated routers, each device in an ad hoc network serves as a router, forwarding packets on behalf of other nodes in the network. Each node therefore requires information regarding available routes to any given destination in the network. This information is acquired through the use of a routing protocol.

The fast growth of wireless networking, combined with the desire for ubiquitous computing, has been a catalyst for research in the field of ad hoc networking. The specific research area of wireless ad hoc routing protocols has been of particular interest, producing numerous protocols characterized over many different classes. A comprehensive overview of existing ad hoc routing protocols and their classifications is provided in [43, 16, 44]. The following section provides an overview of the two main categories of wireless ad hoc routing protocols.

### 2.3.1 Proactive and Reactive Routing Protocols

Proactive and reactive routing protocols generally differ by how a node stores, requests, gathers, and disperses routing information. Proactive routing protocols are generally table driven where each node attempts to maintain routing information about all other nodes at all times. In order to keep routing tables up-to-date, proactive protocols generally require constant communications

between nodes regarding routing/topology information. The advantage of this approach is that routes to a any destination in the network are readily available; the disadvantage is the constant overhead of maintaining routes, even those not being used. An example of a popular proactive routing protocol is the Optimized Link-State Routing (OLSR) [38] protocol.

Reactive (on-demand) protocols generally don't maintain routes to all destinations at all times. Reactive protocols aim to obtain routing information, such as a route to a particular destination, only when it is required. The advantage of a reactive protocol is the lower overhead compared to that of proactive protocols, this is due to less routing control traffic being flood across the network; the disadvantage however, is the delay involved in establishing a route to a particular destination when that route is needed. The Ad-hoc On-demand Distance Vector (AODV) [50] and Dynamic Source Routing (DSR) [39] protocols are two popular reactive protocols.

Apart from wireless ad hoc routing protocols being purely reactive or proactive, hybrid protocols that utilise both proactive and reactive mechanisms do exist. The Zone Routing Protocol (ZRP) [23] specifies the use of a proactive mechanism within a limited zone radius of a node, and a reactive mechanism is only utilised to establish a route to a destination that lies outside the zone radius.

### 2.3.2 Popular Wireless Ad hoc Routing Protocols

Too many wireless ad hoc routing protocols exist to give an overview of each in this section. This section however aims to give an overview of some of the popular and well known routing protocols that exist. The OLSR routing protocol is the main focus of this work, hence, it is not touched on in this section but rather described in detail in section 2.5.

#### *AODV Routing Protocol*

The Ad hoc On-Demand Distance Vector (AODV) routing protocol is an on demand reactive routing protocol designed for ad hoc networks. AODV does not constantly maintain routes to known destinations in the network; however, a route to a particular destination is established on demand only when a route to the intended destination is needed. In general terms AODV is responsible for the following main tasks:

- Route Establishment: If a node using AODV requires a route to a particular destination it will broadcast a Route Request (RReq) packet for that destination; included in the request will be the most recent sequence number the source node maintains of the destination. A recipient node of a RReq will send a Route Reply (RRep) if it is the destination node itself, or if the recipient is an intermediate node which has knowledge of a route to the destination with a sequence number more recent to that in the RReq. Intermediate nodes along the path of a RReq or RRep can also benefit from the route information being transmitted in the exchange.
- Route Maintenance: Owing to the dynamic topology of an ad hoc network and the unpredictable nature of the wireless medium, links between two nodes may become inactive. Route maintenance is the process of detecting missing links and notifying the network of the missing links. When a node senses that a link no longer exists, the node creates

and sends a Route Error (RErr) message in order to notify other nodes that the link no longer exists. Routes maintained by nodes which utilise the unavailable link must then be discarded and new routes discovered.

### *DSR Routing Protocol*

Like AODV, the Dynamic Source Routing (DSR) protocol is reactive protocol and does not rely on constant routing advertisements. It is known as source routing because the source node of a data packet includes the entire route - from source to the packet's destination - in a header of the packet. The source route contains the addresses of each node the packet should traverse, a node receiving the packet forwards the packet to the next hop according to the source route. Intermediate nodes are therefore relieved of routing decisions of a data packet, as all the routing decisions are handled by the source node. DSR consists of two main operations:

1. Route Discovery: DSR nodes maintain a route cache to destinations in the network. If a node intends to send a packet to a particular destination, it first checks the route cache to determine if a route exists. If no route exists, the node initiates a route discovery process and broadcasts a route request packet; a request packet contains the target node's address as well as a unique ID. Intermediate nodes can use the request ID to determine if they have already received/processed this request. If not, an intermediate node appends its address to the request packet and broadcasts it once again. If the target node receives the request, it responds with a route reply containing a copy of the route from the route request.

2. Route Maintenance: Each node forming part of a route between the source and destination of a packet is responsible for performing route maintenance. A node can determine if a link between it and the next hop is bad or no longer exists using several techniques: link-layer acknowledgments; or passive listening to further transmission of the packet at the next hop - using wireless promiscuous mode. If a node determines a link to be unavailable, it must send a route error packet to the source of the data packet that was being transmitted. The source node must then either initiate a new route discovery to the destination or drop the packet. Routes including the unavailable link must then be removed from the route cache.

The DSR protocol also provides various optimisations for the operations listed above. Some of these optimisations include: use of route cache to reply to route requests; caching of routing information overheard by intermediate nodes; piggybacking route reply and route error messages during route discovery process. A more detailed explanation of DSR and its optimisations can be found in [39].

### *The Zone Routing Protocol*

The Zone Routing Protocol is a hybrid routing protocol which relies on both proactive and reactive routing protocols in an attempt to provide efficient and responsive routing in an ad hoc network environment. ZRP is not an actual implementation. Instead, it is a framework specifying the interaction between the proactive and reactive routing protocols involved. Each

node in the network maintains a routing zone around it, this zone is generally specified in terms of number of hops; the zone only includes neighbours that are within the specified hop radius. A proactive routing protocol is used to maintain local routes to all nodes that fall with the zone. When a route is required to a node lying outside the zone, a reactive routing protocol is used to establish a global route to that node. The Zone routing protocol consists of three core components:

- The IntrA-zone Routing Protocol (IARP) [26]: The IARP specifies the proactive routing component. Existing proactive routing protocols can be adapted to work within the ZRP framework.
- The IntEr-zone Routing Protocol (IERP) [25]: The IERP specifies the reactive routing component which is responsible for discovering global routes that lie outside the scope of a particular node's zone.
- The Bordercasting Routing Protocol (BRP) [24]: The BRP is utilized when a global route is required to be discovered. When a route within a particular zone cannot be found, instead of broadcasting a query, the BRP routes the query to specific nodes lying on the border of the zone. In essence, the IERP and BRP work together in utilizing local zone information when discovering global routes. The BRP also ensures the queries are routed away from areas that have already been covered.

**The IntrA-zone Routing Protocol (IARP)**

The IARP is a limited scope proactive routing protocol. It is not a specific protocol but rather a guideline that can be used for existing proactive routing protocols. The IARP in brief terms is a proactive ad hoc routing protocol that operates within what is known as a routing zone. The routing zone R of a node X is specified as a hop-count; if R = 2, node X must only transmit/advertise routing control packets to nodes that fall within 2 hops of X. The transmission of routing material can be limited by the TTL (Time-To-Live) field of a routing packet. If R = 2, the TTL must be initialized to R-1 by the source node. Each intermediate node that receives a routing packet decrements the TTL value. A packet with a TTL = 0 should not be processed and should be discarded.

**The Inter-zone Routing Protocol (IERP)**

The IERP is an adaptation of an existing reactive ad hoc routing protocol. When a route is required by node X to a local node within X's zone, the IERP suppresses the local query as the IARP deals with local routes. When node X requires a route to a destination outside of X's zone, IERP makes use of a bordercasting service to efficiently diffuse the query into the network. The IERP must support IARP routing table imports as well as the ability to perform IARP table lookups; this allows IERP to take advantage of local routes stored by IARP. Therefore when an IERP node Z receives a query for a destination that lies within Z's local zone, Z can reply to the query using local zone information that has been accumulated by the IARP component.

**The Bordercasting Routing Protocol (BRP)**

Bordercasting provides an efficient means to diffuse queries into the network by means of multicasting. When IERP sends a query such as a Route Request, it sends the query to BRP for bordercasting. Instead of simply flooding the network with the query, BRP selectively guides the query toward the peripheral nodes of the zone (peripheral nodes of node X are all nodes that are exactly R hops away from X).

## 2.4 SECURITY IN AD HOC ROUTING PROTOCOLS

Node and route discovery form the core operations of an ad hoc network. Without these tasks, communications in a dynamic ad hoc network would cease to exist. It is a well known fact that ad hoc networks are highly vulnerable from a security point of view [48]. With no infrastructure in place, providing security services such as authentication, access control, confidentiality etc. is a much more challenging task. If security is to be employed in an ad hoc network, it must be an additional task that is enforced by each node. Security is not the only challenge facing ad hoc networks, energy and bandwidth constraints also exist [66]. With these constraints in mind, the majority of existing routing protocols have been designed with efficiency and minimal overhead as primary objectives, thus, often neglecting to incorporate security into the initial design.

A routing protocol for ad hoc networks without any form of security is highly susceptible to many attacks. Without security, each node assumes trust with other neighbouring nodes and blindly accepts routing control data it receives from them. Already one can see the need for security measures, particularly authentication and integrity. A detailed overview of routing attacks is provided in [48]. A brief overview of several well known routing attacks is given below.

### 2.4.1 Wireless Ad hoc Routing Attacks

***Routing Injection Attack***

A routing injection attack occurs when a malicious node injects false routing information into the network. Recipient nodes believe the information to be correct, and update their routing records accordingly to reflect the incorrect information. This form of attack is a direct consequence of an absence of authentication mechanisms.

***Routing Data Manipulation Attack***

A routing manipulation attack is similar to that of an injection attack. However, instead of injecting false routing information, a malicious node simply manipulates the information it receives before forwarding it to the next node. Without any integrity measures the next receiving node will be unable to see any evidence of tampering, and hence, will process the incorrect information.

***Passive Eavesdropping Attack***

The eavesdropping attack is not an active attack like the injection or manipulation attack, but rather a passive one. Eavesdropping occurs when an outside malicious node simply listens in

over the open wireless channel for routing information. From the information it captures, it is able to discover potentially sensitive topology information about the network.

***Packet Dropping Attack***

Another active attack occurs when a malicious node deliberately drops routing packets without forwarding them to the next hop node. This form of attack may prevent certain nodes from receiving up-to-date routing information and hence, prevents them from establishing routes correctly. This attack is a form of denial of service (DoS) attack, and is still an open research challenge.

***Replay Attack***

A replay attack is an active attack which occurs when a malicious node stores routing information and then later retransmits the stored information. Recipient nodes will process the old (incorrect) routing information and subsequently corrupt their routing entries. A digital signature alone is simply not enough to prevent replay attacks as the signature of the replayed (old) information will still verify correctly at the receivers end.

***Black Hole Attack***

The Black hole attack is an active attack that often forms part of a denial of service attack. In a black hole attack, a malicious node uses the routing protocol to advertise a shortest path to a node whose packets it wants to intercept [3]. The node receiving the fake routing announcements will then adopt the routing path through the malicious node. The malicious node will then have access to all data routed along the fake path. The malicious node can then choose to drop all data packets, creating a denial of service.

***Wormhole Attack***

The Wormhole attack [31] occurs when one malicious node tunnels data to another malicious node at which point the data is replayed into the network. The two malicious nodes involved in the attack maintain a separate channel of communications, thus allowing them to communicate directly over long distances. This form of attack is a particular threat to routing protocols. Figure 2.8 shows a scenario representing a wormhole attack: the malicious nodes can tunnel routing control data between them, this tunnel leads nodes A and B to believe they are directly connected, and hence, prevents correct routes being established between them.

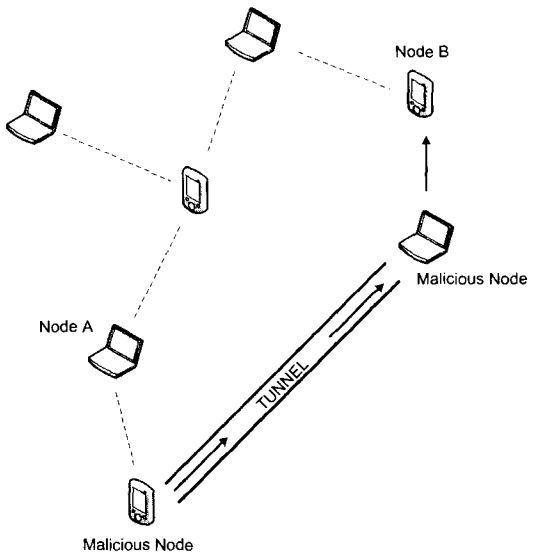



Figure 2.8: The Wormhole attack in a wireless ad hoc network.

### 2.4.2 Summary

In the previous section we have identified, and briefly touched upon, several well known security attacks on wireless ad hoc routing protocols. Table 2.1 below, attempts to categorise these attacks according to the security properties each attack challenges. The purpose of identifying and categorising such attacks is to help in the design process, explained in chapter 3; the design process outlines which security attacks the proposed solution attempts to counter.

| Security Attack | Authenticity | Integrity | Availability | Privacy | State of Routes |
|---|---|---|---|---|---|
| Injection Attack | X | | | | X |
| Manipulation Attack | | X | | | X |
| Eavesdropping Attack | | | | X | |
| Packet Dropping Attack | | | X | | X |
| Replay Attack | | | | | X |
| Black Hole Attack | | | X | | X |
| Wormhole Attack | | | | | X |

Table 2.1: Wireless Ad hoc Routing Attack Summary

## 2.5 THE OPTIMIZED LINK STATE ROUTING PROTOCOL

The Optimized Link-State Routing (OLSR) protocol is a proactive, table-driven link-state routing protocol. OLSR relies on a regular exchange of link-state and neighbour information between nodes in a network in order to keep routing tables up-to-date. The OLSR protocol is a datagram protocol and uses the UDP port 698. This section aims to give an overview of the fundamental operations of the OLSR protocol. For a full description of the protocol, see [11]. The following operations are core to OLSR:

- Link-Sensing and Neighbour Detection: This is the process whereby directly connected nodes discover each other and the status of the links between them. Described in further detail in section 2.5.2.
- MPR Signaling and Selection: MPR (MultiPoint Relay) node selection is a mechanism OLSR uses to make the process of diffusing information throughout the network more efficient. Described in further detail in section 2.5.4.
- Topology Control Message Diffusion: The information gathered from link-sensing and neighbour detection forms the basis for full network topology discovery. Topology information is diffused throughout the network to allow nodes to build routes. Described in further detail in section 2.5.5.
- MID Signaling: OLSR nodes may have multiple network interfaces. If a node is using multiple OLSR interfaces, MID (Multiple Interface Declaration) signaling allows the node to declare these network interfaces in an MID message. In an MID message, the originator header field of the message specifies the main interface address of the originating node, other OLSR network interfaces being used by a node are listed in the message body of an MID message. MID messages are flooded throughout the network at specified intervals.

Owing to the fact that OLSR is a proactive routing protocol, the exchange of routing information amongst nodes is carried out via constant broadcasting of OLSR routing control messages. OLSR uses a generic packet format for the transmission of OLSR control messages. A single OLSR packet can be used to carry multiple OLSR messages and is simply a carrier mechanism over a single hop; the number of OLSR messages that can be carried within a single packet is generally limited by the maximum frame-size of the network. OLSR also specifies a generic message format for each OLSR message. The OLSR generic packet and message format is shown in figure 2.9.

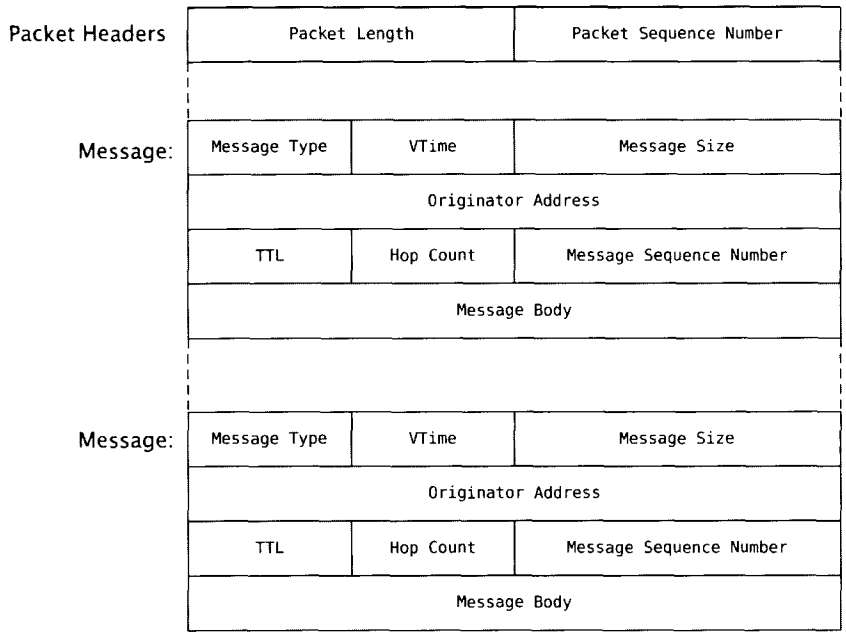



Figure 2.9: The Generic OLSR Packet Format.

The *Packet Length* field simply specifies to size of the entire packet, including all its messages. The *Packet Sequence Number* field is a sequence number corresponding to a network interface, and is incremented for each OLSR packet sent on its corresponding interface. Each message maintains a generic header structure. The message headers provide information regarding the message type being carried. All data that is specific to a message type is carried within the message body field of an OLSR message. The purpose of each message header is explained below:

- Message Type: As it states, this field indicates what type of message is being carried. OLSR specifies several reserve message types for the core functioning of the routing protocol. OLSR however, also allows extended (custom) message types to be defined.
- Vtime: When a node receives a message, the Vtime field indicates how long after reception the contents of the message should be considered valid.
- Message Size: The size of the entire message (including headers) in bytes.
- Originator Address: The IP address of the node this message originated from.
- Time To Live (TTL): The maximum hop-count this message must be allowed to traverse. The TTL of a message is decremented by 1 at each hop prior to being forwarded.
- Hop Count: The current number of hops this message has traversed. The hop-count is incremented by 1 at each hop prior to a message being forwarded.
- Message Sequence Number: The sequence number is a unique identification number for a message. The sequence number of an OLSR node is incremented by 1 for each outgoing message.

### 2.5.1 OLSR Packet Processing

As a general overview, the following steps are taken by an OLSR node when receiving an OLSR packet:

1. The node must process each message within the OLSR packet. For each message, the headers are examined to determine various properties regarding the message in order to determine how to process the message. OLSR requires various checks to be performed on an OLSR message to determine if the message is to be processed, or dropped. Some of these checks include: if the message is a duplicate of a recently received message; if the time-to-live is less than or equal to 0; or if the message originated from this node.
2. If a message is not dropped, it must be processed according to its type. However, it may be the case that an OLSR implementation is unaware of a specific message type it receives. The following steps are taken for both cases:
   - Known Message Type: The message is processed according to its type. Forwarding of the message takes place according to the specifications of the message type.
   - Unknown Message Type: The message is not processed. Forwarding of the message takes place according to the default forwarding policy specified by OLSR.

### 2.5.2 Neighbour Detection and Link Sensing

OLSR discovers the status of immediate links and neighbours through the periodic emission of HELLO messages. HELLO messages are used by an OLSR node to list the current link and neighbour status of each neighbour interface that has been discovered or established. By default, HELLO messages are emitted every 2 seconds and exist only for a single hop for the purposes of link and neighbour detection, thus are never forwarded.

Link-sensing in OLSR is mainly focused on determining if a link with a direct neighbour is symmetric or asymmetric. A wireless link between two nodes is not always bidirectional; owing to the unpredictable and volatile nature of the wireless medium, it may be the case where one node can receive data from a neighbour but not vice versa. If this is the case, the link status is said to be asymmetric. If, on the other hand, two neighbours receive transmissions from one another, the link is said to be symmetric. Consider the following steps explaining a brief overview of link-sensing between 2 OLSR nodes:

1. Two OLSR nodes (node A and B) exist within wireless range of each other and commence broadcasting HELLO messages at their specified emission intervals. At this point, neither A nor B will have discovered any links or neighbours.
2. Node A and B receive HELLO messages from one another, and discover links that exist between them. At the next emission of HELLO messages, node A will list the interface of node B within its HELLO message; at this point the link status of this interface remains asymmetric. Node B does the same with the interface of node A.

3. Using node A as an example, if the subsequent HELLO messages received from B contain the interface of node A, node A can infer that node B has received its HELLO messages, and hence the link can be established as symmetric. Should node B stop receiving HELLO messages from A, this will eventually result in the absence of node A's interface within B's HELLO messages. Node A will realise that communications are only taking place in one direction, and the link will eventually become asymmetric once again.

An OLSR node must also maintain a neighbour set that is directly associated to the link-set of the node. OLSR classifies a node as a neighbour if at least one link exists with the node. Associated with each neighbour is a status which reflects the current relationship with the neighbour. For each neighbour interface entry listed in a HELLO message, an OLSR node must also disclose the neighbour status of the interface. Apart from a neighbour set, a node must also maintain what is known as a 2-hop neighbour set. A node is classified as a 2-hop neighbour if: 1) it is not a 1-hop neighbour, and 2) it maintains a symmetric link to a symmetric 1-hop neighbour. One can see that if a node X broadcasts a HELLO message containing the status of its neighbours, a direct neighbour node Y can determine the symmetric neighbours of X, and hence any 2-hop neighbours.

### 2.5.3 OLSR Link Hysteresis

In a wireless ad hoc network, it may be the case where two nodes can hear each other but the link between them is either very bad or temporary. For instance: two nodes lie just within each others wireless range but there is significant packet loss; or, in a mobile context, two neighbours can hear each other's transmissions, but only briefly as one node moves in and out of wireless range. In the context of OLSR, this could lead to poor links being formed. OLSR uses a strategy known as link hysteresis to prevent these poor links forming. Below is a brief explanation of the link hysteresis strategy.

A link quality value (ranging between 0 and 1) is associated with each neighbour interface maintained by an OLSR node. The link quality associated with a neighbour interface is dependent on the successful reception of OLSR packets on that interface. If an OLSR packet is successfully received, the link quality is scaled upward as follows:

$$link\_quality = (1 - HYST\_SCALING) * link\_quality + HYST\_SCALING$$

where $HYST\_SCALING$ is a scaling OLSR constant. If an OLSR packet loss is detected on a neighbour interface, the following instability rule is applied to the link quality:

$$link\_quality = (1 - HYST\_SCALING) * link\_quality$$

OLSR packet loss is detected through the monitoring of packet sequence numbers, and by a specific period of silence on a neighbour interface. A period of silence is detected through the expiration of a timestamp, created at the reception of the last successful HELLO message. The

timestamp interval is set to the expected emission interval (Htime field) of a HELLO message on the network interface in question.

Threshold values (HYST_THRESHOLD_HIGH, HYST_THRESHOLD_LOW) are used to determine if a neighbour link is established or lost, based on the value of the link quality. If the link quality drops below HYST_THRESHOLD_LOW, the link becomes lost. If the link quality rises above HYST_THRESHOLD_HIGH, the link is established as stable. Whilst it may take longer to establish a stable link using hysteresis, hysteresis makes neighbour links more robust against sporadic packet loss and reception.

### 2.5.4 Multipoint Relay Selection

Multipoint relay nodes are special nodes which have two additional responsibilities: 1) generation of Topology Control (TC) messages; and 2) forwarding of OLSR messages. The MPR mechanism employed by OLSR reduces the transmissions required to flood the network with routing data. Instead of every node simply relaying every message, only MPR nodes relay messages that are intended to be forwarded. The MPR mechanism makes the process of network flooding more scalable and efficient [37].

Link-sensing and neighbour detection form the basis for multipoint relay selection. Once an OLSR node has performed link and neighbour detection, and has established a 2-hop neighbour set, it must select symmetric neighbours as MPR nodes such that every 2-hop neighbour can be reached via the MPR nodes. To select a symmetric neighbour as MPR, a node must signal the neighbour as an MPR by indicating an MPR status as the neighbour's status in subsequent HELLO messages. Once a node has been selected as an MPR node by a neighbour (known as the MPR Selector), it has the responsibility of forwarding OLSR messages it receives from the MPR-selector's interface. Nodes that have not been selected as MPR nodes, must not forward OLSR messages. Figure 2.10 depicts network flooding using the MPR mechanism.

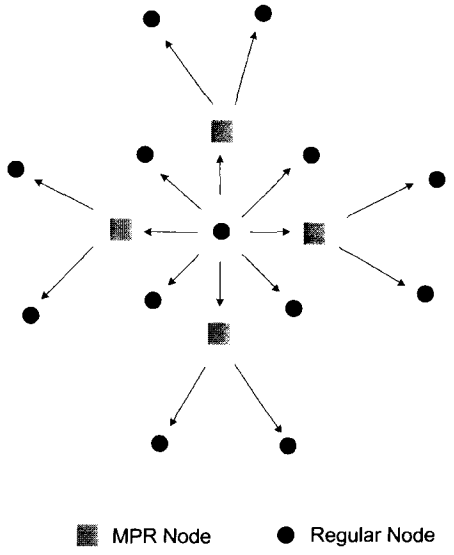



Figure 2.10: OLSR MPR flooding mechanism.

### 2.5.5 Topology Control Messages

OLSR relies on the use of Topology Control (TC) messages to distribute information about neighbours throughout the entire network. The IETF OLSR draft [11] specifies that only nodes that have been selected as MPR nodes must generate and broadcast TC messages at the specified TC interval. Using the knowledge of which nodes are accessible, via the MPR nodes distributing the TC messages, recipient nodes can calculate multi-hop routes to the nodes advertised in the TC messages. By default, TC messages are generated and emitted every 5 seconds.

The OLSR protocol specifies a TC_REDUNDANCY parameter which governs the detail of information that is to be transmitted in a TC message; possible parameter values include:

- 0: TC messages contain only neighbours which belong in the MPR Selector set. This is the OLSR default value.
- 1: TC messages contain neighbours that belong in either the MPR set or the MPR Selector set, or both.
- 2: TC messages contain the full neighbour set.

## 2.6 Security for the OLSR Routing Protocol

### 2.6.1 Lower-Layer Security

Whilst a lower-layer security mechanism, such as IPSec [41], may be sufficient in securing user data, it is not always applicable when it comes to securing the routing data in an ad hoc network [67]. In [67], Zapata et al. explain that user data is generally point-to-point, and therefore can be secured using a point-to-point security mechanism such as IPSec. On the other hand, routing data is typically not point-to-point, but rather broadcast data which is processed at each hop, possibly modified and broadcast once again. In the context of OLSR, packets change on a hop-by-hop basis; for instance: a packet may be carrying two messages (TC, HELLO), the TC message may be forwarded in a new packet at the next hop, however the HELLO message will not be forwarded. Hence, IPSec cannot be relied on for securing OLSR data at each hop from source to every intended destination.

The use of MAC-layer (Medium Access Control layer) security, such as 802.11i, will also not produce the desired security; MAC-layer security provides a secure link for a single hop. Consider the scenario shown in figure 2.11: the use of MAC-layer security only provides node A a security association with its direct neighbour node B. Node B has the opportunity to maliciously modify the data prior to resending it to node C. There is no security association created between end nodes A and C.

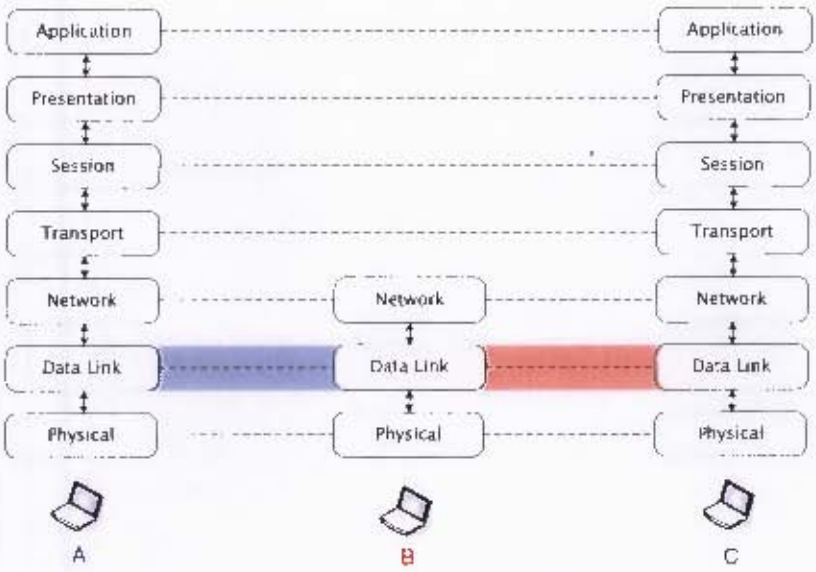



Figure 2.11: MAC-layer security in a multi-hop network environment.

### 2.6.2 End-to-End Security for the OLSR Routing Protocol

With regard to data security in an ad hoc network, the term *end-to-end security* concerns itself with ensuring that some defined security condition, of the data in question, is satisfied from the source to the destination of the data. End-to-end security of OLSR routing data would require that the routing data is secure, even if this information is routed via non-trusted nodes over multiple hops. As it has been shown above, lower-layer security mechanisms, such as IPSec and MAC-layer security, will not provide the desired end-to-end security required for the routing data of OLSR. Instead, the security mechanisms need to be specific to OLSR, and hence, implemented at an application level.

#### *Application-Level Security*

Now that it has been established that application-level security is needed, is it suitable to simply apply security mechanisms to just the OLSR packet, and not individually to each message within the packet? Like MAC-layer security, this approach will only provide security for a single hop; however, it is suitable for use in an environment where the ad hoc network is comprised of trusted nodes. Using this approach, the security is present to protect the routing data from outside attacks on a hop-by-hop basis. Unlike an OLSR packet, which is simply a carrier mechanism for OLSR messages and will only exist for a single hop, a message may traverse multiple hops from source to each intended destination. Hence, securing the OLSR message should be the primary focus if end-to-end security of the routing data is desired. Figure 2.12 shows the same scenario as in figure 2.11, however security is being applied to each message at the application-level.

Figure 2.12 shows that when security is applied specifically to the OLSR application, an end-to-end security association is created between originator and destination - as is the case with node A and C. The security mechanisms applied to a message, allow a recipient to perform

security checks on the message and its originator, irrespective of how many hops the two nodes are apart.

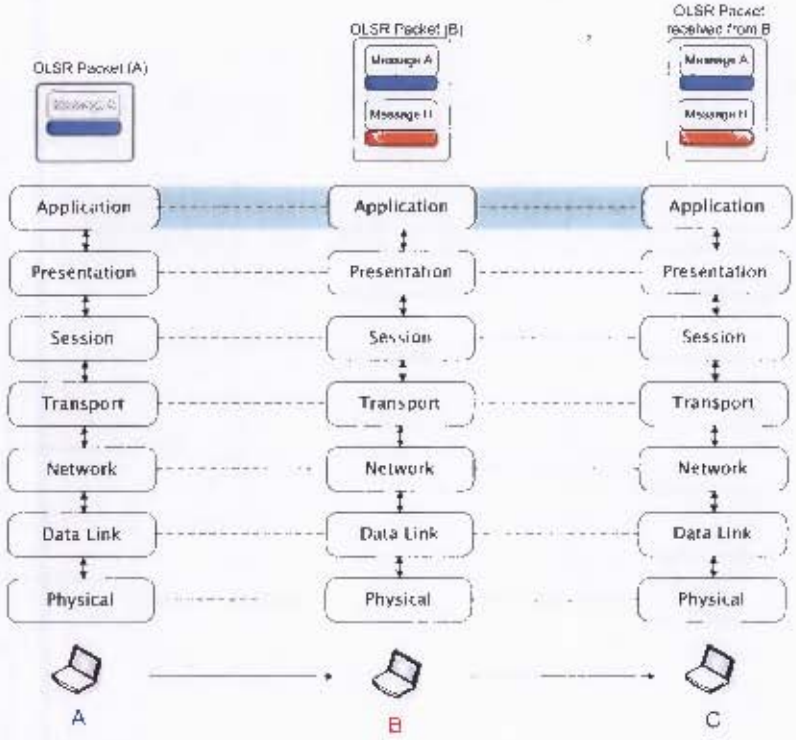



Figure 2.12: Application-level security in a multi-hop network environment.

### 2.6.3 Existing Secure OLSR Schemes

Several secure schemes for OLSR have been proposed in other works [30, 12, 27]. The OLSR security scheme presented in [27] provides signature and timestamp protection over an OLSR packet, and thus only provides hop-by-hop security. A signature is created using a shared symmetric key, and is carried along with a timestamp as a separate OLSR message in a packet. A loose time synchronisation, between immediate neighbour nodes, is carried out using a timestamp request/response mechanism. The scheme is compatible and able to interoperate with the standard OLSR protocol.

The security scheme presented in [12] uses digital signatures and timestamps to protect the content of each OLSR message. The digital signature and timestamp are carried in a separate message within the same packet as the OLSR message they are protecting. The security scheme proposed in [12] also provides mechanisms for dealing with misbehaving or compromised nodes in an ad hoc network.

In [30], an OLSR security scheme known as SOLSR is proposed. SOLSR provides protection of routing data using digital signatures. Mutable fields, such as time-to-live and hop-count fields are also protected using a hash-chain mechanism. SOLSR proposes a wormhole detection mechanism for the OLSR protocol.

## 2.7 SECURITY PRIMITIVES

### 2.7.1 Asymmetric Digital Signatures

In general terms, a digital signature is a message digest that has been secured using an encryption key. A digital signature provides a means to: 1) authenticate, and 2) verify the integrity of the data it represents. Asymmetric digital signatures require the use of asymmetric key pairs. Unlike symmetric signatures, which require the same key in the creation and verification process of a signature, asymmetric signatures use two specially correlated keys known as a public and a private key; if one key is used to create a signature, the other key has to be used to verify the signature.

#### *Identity Based Signature Schemes*

Traditionally, a private key is used to create a digital signature and the corresponding public key is used to verify the signature. Prior to explaining the concept of an Identity Based Signature (IBS) scheme, an overview of RSA [55], a well known public key system, is provided. Stallings [58] describes the various steps of RSA key generation, encryption, and decryption as:

**RSA Public/Private Key-pair Generation**

The RSA key-pair generation can be explained in the following steps:

1. Select two large random prime numbers $p$ and $q$
2. Calculate $n = pq$, known as the RSA modulus
3. Determine the value $\phi = (p-1)(q-1)$
4. Select a prime number $e$ which is relatively prime to $\phi$
5. Calculate $d = 1/e \bmod \phi$
6. Public Key = $(e, n)$. Private Key = $(d, n)$

**RSA Encryption**

In order to encrypt a message M with user A's public key, the following procedure is used:

$$C = M^e \pmod{n}$$

with C being the resulting ciphertext, and the pair (e, n) being A's public key. For user A to create a digital signature of a message M, the following steps can be taken:

1. Create a hash of the message $H(M)$, using a secure one-way hash function.

2. Use an RSA private key to encrypt $H(M)$:

$$C = H(M)^d \pmod{n}$$

where the pair (d, n) is the private key of user A.

**RSA Decryption**

For user A to obtain the plaintext M of a message C that has been encrypted using A's public key:

$$M = C^d \pmod{n}$$

where (d, n) is A's private key. User B can verify a digital signature of a message M that has been created by user A as follows:

1. Create a hash of the message $H'(M)$, using a secure one-way hash function.
2. Decrypt the digital signature C attached to M using A's public key (e, n) to obtain the original hash $H(M)$:

$$H(M) = C^e \pmod{n}$$

3. Verify that $H'(M) = H(M)$

The security of the RSA public key algorithm comes from the difficulty associated with factoring a large integer $n$ into its two prime factors [58]. The RSA algorithm typically relies on a trapdoor one-way function for its security, where:

1. Given the values $e$ and $d$, it is easy to calculate the product $n$.
2. Given the values $e$ and $n$ (public key components), it is computationally infeasible to derive the value $d$.

As one can see from the explanation of RSA above, a user can only verify a digital signature if it is in possession of the creator's public key. Associated with this requirement is the task of public key distribution, i.e. the distribution of a user's public key to other users in the network in a secure manner. In an environment such as an ad hoc network, where a central key authority cannot be assumed, the task of public key distribution is non-trivial. An Identity Based Signature scheme differs from a traditional public key scheme, such as RSA. An IBS scheme requires

that publicly known information about a user, such as a user's IP or E-mail address, be used to derive the public key of a user. Therefore, public key distribution is not a problem, as a user's public key can be derived from information that is accessible to everyone.

In [7], Baek et al. explain the basic concepts of an IBS scheme as:

1. Setup: A trusted third party creates a master public and private key denoted as $\text{pk}_{PKG}$ and $\text{sk}_{PKG}$ respectively. It is required that the $\text{pk}_{PKG}$ is distributed to all parties involved in the network; the distribution of $\text{pk}_{PKG}$ can be achieved in various ways.
2. Private Key Extraction: The signer bob authenticates to the trusted party and obtains a private key $\text{sk}_{IDbob}$ which is derived using the signers identity $\text{id}_{bob}$ and the master secret key $\text{sk}_{PKG}$.
3. Signature Generation: Using the private key $\text{sk}_{IDbob}$, bob creates a signature $\sigma$ over a message m.
4. Signature Verification: Having received $\sigma$ and m, the recipient uses the identity of the sender $\text{id}_{bob}$ and the master public key $\text{pk}_{PKG}$, to verify the signature of the message m.

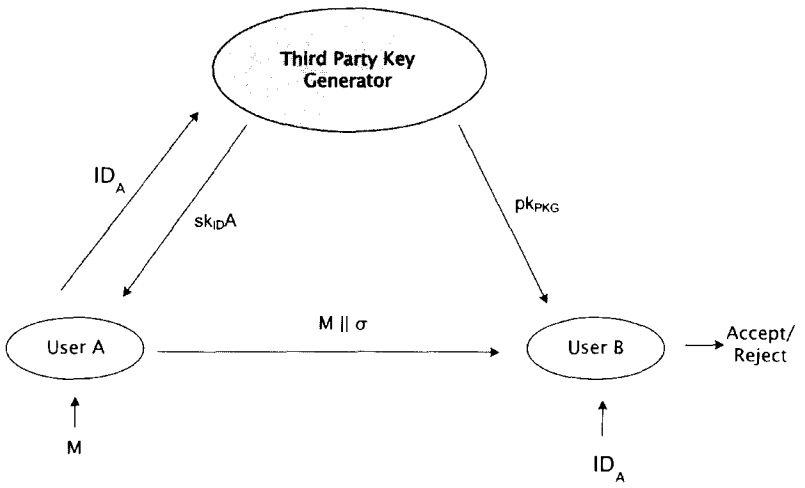



Figure 2.13: Identity-based signature system.

Shamir [56] constructed an IBS scheme that can be realised using an existing RSA function. The key construction, signature creation, and verification procedures of Shamir's identity-based signature scheme are explained below:

**Shamir's IBS Key Construction**

1. Create a master RSA public/private key pair. The public key is $(e, n)$ and the private key is $(d, n)$.

2. Create an identity-based public key $ID$. $ID$ is simply some publicly known information, such as an e-mail address or possibly a hash of an IP address.

3. Create an identity-based private key $s_{ID}$. The identity-based public key along with the master private key is used in the creation of the private key:

$$s_{ID} = ID^d \pmod{n}$$

**Shamir's IBS Signature Creation**

To sign a message $m$:

1. Create a random number $r$ and compute

$$t = r^e \pmod{n}$$

2. Use a secure one-way hash function H to compute $f$, and subsequently to compute $s$ as:

$$f = H(t, m)$$

$$s = s_{ID} \cdot r^f \pmod{n}$$

3. The signature of $m$ is then the pair $(s, t)$.

**Shamir's IBS Verification Condition**

To verify the signature $(s, t)$ of a message $m$:

1. Verify that the following condition holds:

$$s^e = ID \cdot t^{H(t,m)} \pmod{n}$$

**Problems with Identity-Based Signature Schemes**

Whilst it is easy to see the advantage of using an IBS scheme with regard to public key distribution, there is a weakness with the identity-based signature system shown in figure 2.13. This system suffers from a key escrow property [7]. A third party constructs a user's private key using a master private key. As a result, the third party is able to use any user's private key for signature purposes of any messages. This property essentially removes the non-repudiation offered by a user's signature. Non-repudiation essentially offers a property in the case of two entities exchanging information, whereby neither entity can subsequently deny their participation in the exchange [13].

### *One-Time Signature Schemes*

In many circumstances, the time and processing penalty incurred by using traditional public key digital signatures, such as RSA, may be too costly to the system or program being used [46]. Embedded systems and sensor networks are typical examples of this. One-time signatures is an area of research which aims to find alternative methods of providing asymmetric cryptographic properties. One-time signature schemes are generally based upon the use of one-way hash functions, and thus aim to offer better performance when compared to traditional public key signature schemes based on trapdoor functions. Whilst from a performance point of view, one-time signature schemes offer an advantage, these schemes generally also incur limitations and additional complexities; a one-time signature scheme is generally limited to signing one, or in some cases, a small number of messages per key-pair. This limitation may invoke regular key refreshes. One-time signatures also present complexities with key establishment/distribution, and generally incur large key and signature sizes [46].

One-time signatures were originally proposed by Lamport [42]; recently however, several new schemes have been proposed. The scheme proposed in [51], known as the BiBa signature scheme, features small verification overhead and small signature sizes with the downside of large public key sizes and high signature generation overhead. In [53], Reyzin et al. improve on the BiBa scheme by introducing a simple scheme that offers faster signature creation and verification times; this scheme is known as Hash to Obtain Random Subset (HORS). The key generation, signature creation, and signature verification procedures of HORS are explained below:

#### Key Pair Generation

In order to sign messages of a broadcast nature, a node must first generate its own public and private key pair. The procedure to generate this key pair is described below.

1. Given: Parameters t, k, $l$.
2. Create t random $l$-bit strings $s_1, s_2, \ldots, s_t$.
3. let $v_i = f(s_i)$ where f is a one-way function on an $l$-bit input string.
4. Public Key $PK = \{k, v_1, v_2, \ldots, v_t\}$
5. Private Key $SK = \{k, s_1, s_2, \ldots, s_t\}$

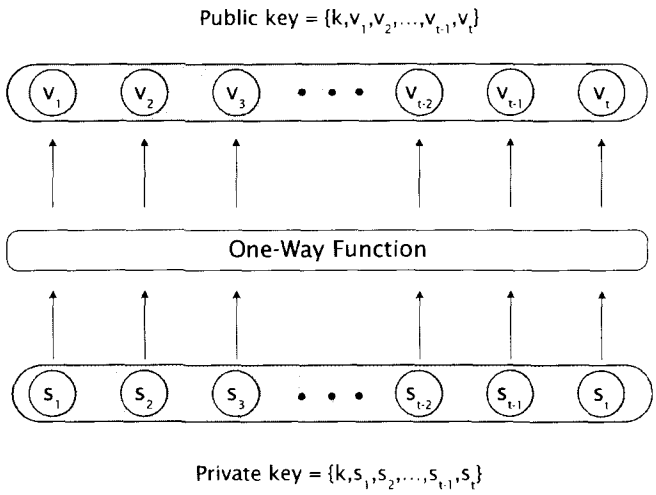



Figure 2.14: Generation of a broadcast HORS public/private key pair.

**Signature Creation**

The procedure for signing a message M is as follows:

1. For a message M, create h = Hash(M). A cryptographic hash function such as SHA-1 [47] can be used.
2. The hash of the message h is then split into k pieces of length $\log_2 t$ bits. Each piece is then interpreted as an integer $i_j$ written in binary for $1 \leq j \leq k$.
3. Combine the integers to form a subset of T of size at most k.
4. Each integer $i_j$ is then used as an index for a private value $\{s_1, s_2, ..., s_t\}$ from the private key.
5. Signature of M = $\{s_{i1}, s_{i2}, ..., s_{ik}\}$.

**Signature Verification**

To verify a signature, a recipient will require the corresponding HORS public key of the sender. The procedure for verifying a signature is as follows:

Given a message M, signature of M = $\{s'_1, s'_2, ..., s'_k\}$, and public key of sender = $\{k, v_1, v_2, ..., v_t\}$

1. Calculate h = Hash(M). Split h into k pieces of length $\log_2 t$.
2. Interpret each piece as an integer $i_j$ written in binary for $1 \leq j \leq k$.
3. Verify that for each integer $i_j$, $f(s'_j) = v_{ij}$, where f is a one-way function.

**HORS Security Parameters**

Whilst HORS provides efficient signature operations, it offers lower security when compared to traditional asymmetric signature algorithms such as RSA. HORS offers a trade-off between the level of security that is provided, and the number of signatures a single key pair is limited in producing. The size of a HORS key, coupled with the the number of private values disclosed in each signature, also affect the strength of security offered by HORS.

- The parameter t specifies the number of hash values that exist in a public and private key, t therefore impacts most on the size of a HORS public and private key pair.
- Parameter k specifies the number of values from a private key that must be included in a signature. Parameter k impacts most on the size and processing requirements of a HORS signature.
- The parameter $r$ specifies the maximum number of signatures a private key may provide; the higher the value of r (with all else being constant), the weaker the security offered.

In [53], Reyzin et al. define the security strength (in bits) offered by the HORS algorithm as:

$$k(\log_2 t - \log_2 k - \log_2 r)$$

Reyzin et al. [53] suggest the following values for the parameters t, k, and l:

1. $\{t, k, l\} = \{256, 20, 80\}$

$$Public\ key\ size = t \cdot hash(l) = 256 \cdot 160 = 40960\ bits = 5120\ bytes$$

$$Private\ key\ size = t \cdot l = 256 \cdot 80 = 20480\ bits = 2560\ bytes$$

$$Signature\ size = k \cdot l = 20 \cdot 80 = 1600\ bits = 200\ bytes$$

2. $\{t, k, l\} = \{1024, 16, 80\}$

$$Public\ key\ size = t \cdot hash(l) = 1024 \cdot 160 = 163840\ bits = 20480\ bytes$$

$$Private\ key\ size = t \cdot l = 1024 \cdot 80 = 81920\ bits = 10240\ bytes$$

$$Signature\ size = k \cdot l = 16 \cdot 80 = 1280\ bits = 160\ bytes$$

Parameter set 1 offers smaller key sizes and larger signatures when compared with parameter set 2. Although parameter set 2 requires very large key sizes, it does offer a higher level of security for a given value of $r$. Figure 2.15 shows the security offered by the HORS signature scheme using the two parameter sets listed above, whilst varying the value of $r$. The graph in figure 2.15 has been plotted using the HORS security equation, from the previous page, for different values of $r$.

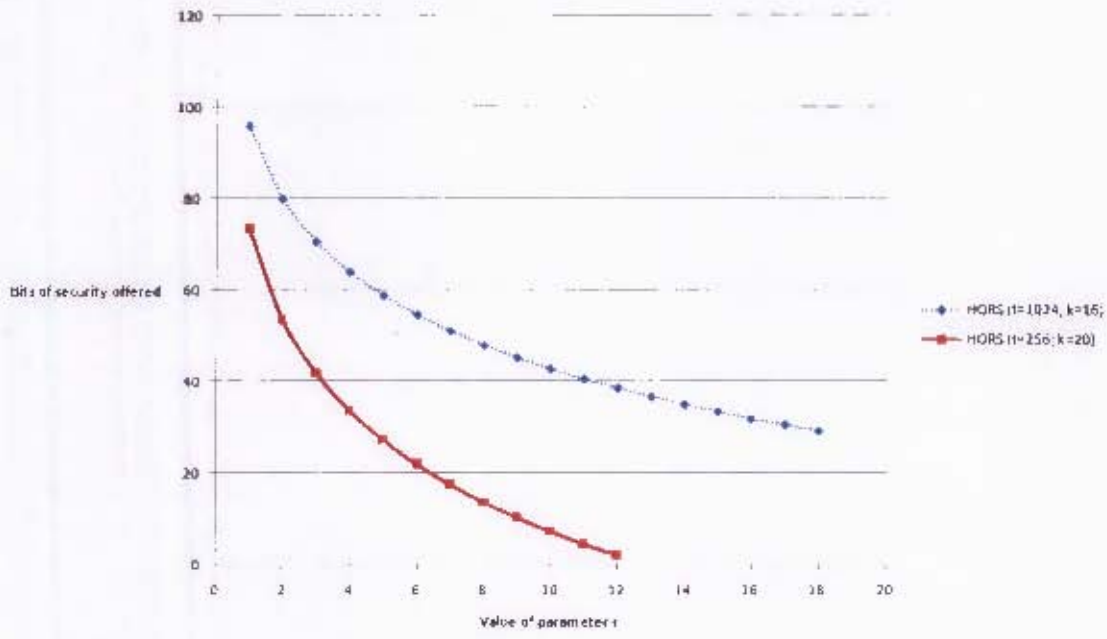



Figure 2.15: Security offered by HORS based on parameter r.

For a given value of $r$, parameter set 2 offers higher levels of security, this is due to the larger number of values $t$, in a public/private key, and the smaller number of values $k$, disclosed in a signature. It must be noted, that when comparing the use of the different parameter sets, a value for $r$ must be chosen for each set, such that the security offered by both sets of parameters is roughly the same. For the remainder of this dissertation, we denote parameter set 1 as *HORS 256* and parameter set 2 as *HORS 1024*.

## 2.8 Conclusion

In this chapter, we have introduced the field of wireless networking, and discussed a decentralised wireless networking model known as ad hoc networking. Wireless ad hoc routing protocols have been introduced, and the importance of the correct functioning of such protocols, in permitting multi-hop communications, has been raised. The emphasis of the chapter has been on the security surrounding routing protocols for ad hoc networks. Security threats facing ad hoc routing protocols were identified and discussed, and the need for end-to-end security associations, between entities communicating over multi-hop channels, was emphasised. The Optimized Link State Routing protocol was examined in detail, as it forms the major focus of the work done in this dissertation. Two signature schemes, differing substantially in nature, were

CHAPTER 3

# SECURITY DESIGN FOR THE OLSR PROTOCOL

## 3.1 DESIGN ASSUMPTIONS AND LIMITATIONS

The design and implementation done in this work is engineered around a wireless testbed (see section 5) that is used in this work for experimental purposes. The testbed serves as a platform to allow specific security mechanisms to be tested with OLSR.

Whilst the testbed environment allows empirical results to be gathered, it must be stated that the results obtained are only representative of a testbed comprised of homogeneous wireless nodes. Over the course of this study, it was not possible to perform experiments with heterogeneous wireless devices such as cellular telephones, PDAs (Personal Digital Assitants), mobile notebooks, etc. The inability to perform experiments in such a heterogeneous environment will serve as a limitation on the generality of the secure OLSR protocol tested in this study. Barring this limitation, the testbed still provides a controlled environment in which to study the feasibility of adding security mechanisms to the base OLSR routing protocol.

Regarding the design and use of security mechanisms with the OLSR protocol, several assumptions were made and these are discussed below:

### 3.1.1 Key Information

Security key management in ad hoc networks is a non-trivial task and is still very much an active area of research. Neither the use of a key management scheme, or the presence of a central key server is considered in this work. It is assumed that the necessary security keys are pre-shared on the routing nodes. The following pre-shared keys are required on each node:

- Master RSA Public key.
- Identity-Based Private Key based on Shamir's construction [56].
- AES Symmetric Encryption/Decryption Key.

### 3.1.2 **Homogeneous Security Parameters**

The wireless test-bed used in this work consists of homogeneous wireless nodes that form a multi-hop environment. To test the performance impacts of the security when using different parameters and algorithms, the wireless nodes are executed using the same parameters and the same security algorithms for any one execution run. In a more realistic environment, different parameters and algorithms may have been used, however it was not necessary for the testing done in this work.

## 3.2 THE OLSR SECURITY DESIGN OVERVIEW

Here a design is constructed, and centered around providing the following security features to the OLSR protocol:

- End-to-End Message Authentication: Routing control messages may travel over several hops, it is therefore essential that any received messages can be authenticated by the recipient node.
- End-to-End Message Integrity: Integrity checks provide a mechanism for a recipient node to ensure that the received data is in its original state and has not been manipulated or altered.
- Message Replay Protection: Prevent OLSR messages from being replayed, therefore preventing the recipients of these messages from processing old routing information.
- Protection of Message Mutable Fields: A mechanism to provide protection of the mutable fields, of an OLSR message, from being altered in a negative manner.
- Routing Data Confidentiality: The confidentiality of routing control data is not as important as its integrity and authenticity. However, it does prevent an outside adversary from passively monitoring routing traffic and obtaining potentially sensitive information about a network topology.

### 3.2.1 Security-Aware OLSR Message

Several mechanisms are required in order to achieve the security features listed above. The OLSR message is the most fundamental building block of the OLSR protocol. Figure 3.1 shows the general format of a security-aware OLSR routing message used throughout this work.

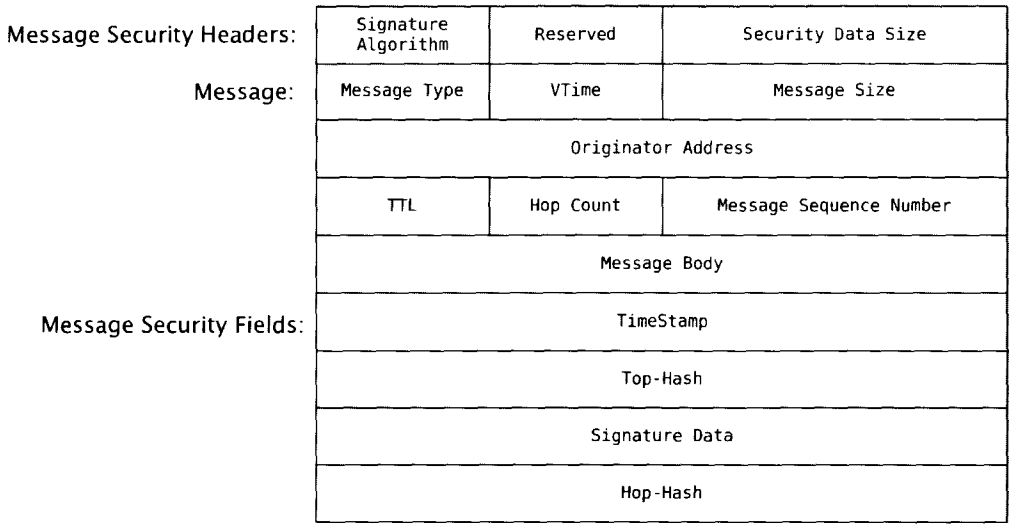



Figure 3.1: Security-aware OLSR message.

The security fields represent the following:

- Signature Algorithm: Specifies which signature algorithm has been used to create a signature of this message.
- Reserved: For later use, also for 32-bit word alignment.
- Security Data Size: Specifies the size of the appended security data (timestamp, top-hash, signature, Hop-hash) in bytes.
- Timestamp: A timestamp is used to provide protection against replay attacks of old routing messages.
- Top-Hash and Hop-Hash: These two hash fields are used to provide a form of protection for the mutable fields of an OLSR routing message, which change on a hop-by-hop basis. The use of such hash fields to protect mutable data is proposed in [68]. The protection of mutable fields is discussed in greater detail in section 3.5.
- Digital Signature: The digital signature to provide message authentication and integrity.

This design requires that all OLSR messages, reserved or extended types, be secured using the format shown in figure 3.1, unless otherwise explicitly specified.

## 3.3 DATA AUTHENTICATION AND INTEGRITY

When securing an OLSR message using the security-aware format of figure 3.1, the signature is created over the following fields:

- OLSR security headers
- Standard OLSR message headers
- OLSR message body
- Timestamp
- Top-Hash field

The Hop-hash field is not included in the signature as it changes from hop-to-hop. More information regarding the use of the top-hash and hop-hash fields to protect mutable fields is disclosed in section 3.5.

Asymmetric signatures have already been discussed in section 2.7.1; prior to discussing their use in this work, the shortfalls of using symmetric key signatures, in the context of OLSR, is explained. If a symmetric encryption/decryption key is used to create a digital signature, all communicating parties must share the same symmetric key.

One can see that the use of a symmetric key signature would not provide end-to-end authentication and integrity of an OLSR message traversing several hops. Figure 3.2 shows this scenario.

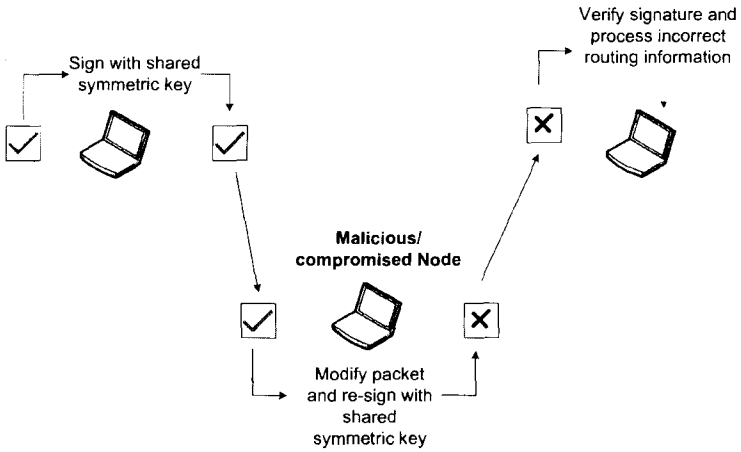



Figure 3.2: End-to-End problem with symmetric keys.

The figure shows a situation where a node generates an OLSR control message, signs the message with a symmetric key, and broadcasts it. At the next hop the node receives the message, verifies the message, and if successful, processes it. If intended, the node should relay the message to the next hop. However, what if the node has a malicious intent? Because symmetric keys are used, the intermediate node has the freedom to change the contents of the original message, sign the modified message with the shared symmetric key, and then relay the altered message to the next hop. The recipient at the next hop then follows the same procedure: verify the message with the shared symmetric key, and if verified, process the message. The signature however would verify correctly, even though the original contents had been modified. After all, there is no evidence to suggest any modifications took place.

The example above emphasises the point that end-to-end authentication and integrity, in the OLSR context, cannot be provided by symmetric key signatures. Hence, the use of asymmetric digital signatures is required. Looking at the same scenario mentioned above, with asymmetric keys: a node generates a control message and signs it with its private key. An intermediate node can verify the message with the sender's public key. However, the intermediate node cannot modify the message and reproduce a signature without knowing the sender's private key. Hence, any modification will be noticed by a node at the next hop and the information will be discarded.

This section has provided a general overview of the use of asymmetric signatures in the OLSR protocol. Section 3.7 provides a detailed overview of the specific signature schemes used in this design, and how they are incorporated into the OLSR protocol.

## 3.4 REPLAY ATTACK PROTECTION

Whilst a digital signature may provide authentication and integrity of a routing message, which is very important, a single digital signature cannot alone provide all the protection needed against attacks [63]. The replay attack, is one attack which cannot be prevented simply with the use of a digital signature. To prevent a replay attack occurring, a timestamp mechanism for each routing message is required. A message timestamp is basically a piece of data which is an accurate reflection of the senders time when the message was created. This allows a receiver to judge if a message is old or being replayed. The method of employing a timestamp mechanism in a protocol differs greatly depending on whether the clocks of the communicating entities are synchronized or not. The two different approaches are explained in this section.

### 3.4.1 Synchronized Clocks

If it is assumed that the clocks of the communicating entities are synchronized, then the task of employing a timestamp mechanism is fairly trivial. As an example, consider a case of the OLSR routing protocol where each message has a timestamp associated with it. If all nodes had synchronized clocks, a node receiving a message could compare the timestamp of the message with its current time. Provided the difference of the two times is not greater than some pre-defined threshold or slack, the message is accepted. A replay attack would thus be difficult, as a replayed message would be identified as old or stale information by a recipient node.

### 3.4.2 Unsynchronised Clocks

In a situation where the clocks of the communicating entities are not synchronized, a timestamp mechanism becomes less trivial. Consider the same case as above with unsynchronised clocks: when a node receives a control message, it can calculate the difference between the timestamp of the message time and its current time. However, because the clocks are not synchronized, this calculated difference is meaningless. A recipient node has no relative time value to compare against the time of the message. Before a recipient node can verify a timestamp, it needs to have some form of knowledge regarding the sender's time.

### 3.4.3 OLSR Timestamp Mechanism

This design assumes no clock synchronisation between OLSR nodes. Although unsynchronised clocks provide additional difficulties when dealing with timestamps between communicating entities, assuming that clocks are synchronised in an ad hoc network environment is not always a realistic assumption. Before a recipient of a message can verify the timestamp of the message, it requires some knowledge of the sender's current time. Here, a recipient node is required to know an upper-bound of the sender's current time as specified in [52]. Figure 3.3 shows a loose time synchronisation protocol presented in [52]. The loose time synchronization protocol consists of the following steps:

1. Node A receives a message from node B. A has no time information for node B, therefore initiates the loose time synchronisation protocol with B. A sends a request at time $T_R$.

2. B receives the request at its time $T_S$. B includes its current time in a response message and sends the response back to A. Ideally, node A would like to know its current time at time $T_S$. This would allow node A to calculate the real synchronization error $\delta$ between A and B. However, node A cannot determine the exact transmission delay of the request message and hence cannot determine its current time at $T_S$.

3. Upon receiving the response from node B, node A stores the time $T_S$ from the response, as well as $T_R$. At node A's current time $T_r$, the round-trip time (RTT) $\Delta$ can be calculated as the time difference between $T_R$ and $T_r$. Node A assumes $\Delta$ as the maximum time synchronization error between itself and B.

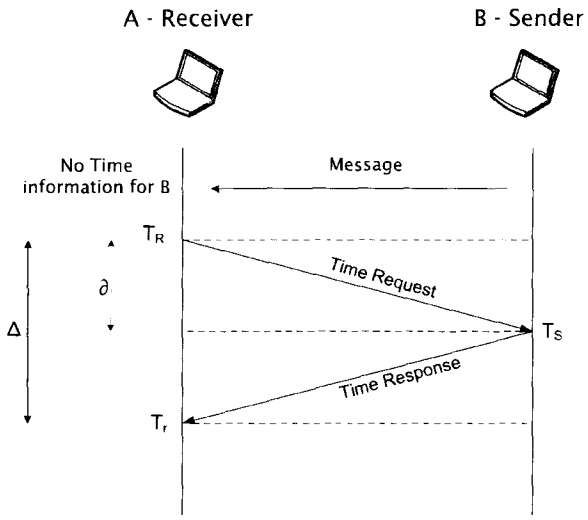



Figure 3.3: Loose Time Synchronisation Protocol.

Upon receiving a future message, a recipient node can calculate an upper-bound time $T_s$ of a sender as specified in [52] as:

$$T_s \leq T_r - T_R + T_S$$

where $T_r$ is the receivers local time at the time of reception, $T_R$ and $T_S$ are the values stored from the loose synchronisation exchange. Once the upper-bound has been calculated, the upper-bound and $\Delta$ values are used to determine if the message timestamp $T_{message}$ falls within an allowed window period:

$$T_{message} \geq T_s - (\Delta + slack)$$

If the conditions above do not hold for a given message, the message is considered old and is discarded. Routes between nodes however, may change on a regular basis in an ad hoc network, thus the RTT ($\Delta$) calculated between two nodes may become irrelevant. Also, the RTT calculated in a loose time synchronisation exchange may not always be accurate in a once off

exchange, and hence, perform badly: for instance, an RTT that is too low may result in too many failed verifications, resulting in too many dropped messages. To overcome this, two mechanisms are used: 1) if the hop-count metric of a route between a pair of nodes changed, a resynchronisation takes place; and 2) threshold values $n$ and $m$ are used to trigger a resynchronisation - in this context we mean loose time resynchronisation. If $n$ messages out of the last $m$ received messages from the same originator have produced invalid timestamp verifications, then a resynchronisation exchange with the message originator is initiated. If a resynchronisation takes place due to this threshold being met, provided the RTT from the resynchronisation is larger than the current RTT, an average of the RTT from the resynchronisation and the old RTT is calculated and stored as the new RTT.

The timestamp mechanism is in place to prevent replay attacks of incoming messages. However, what prevents an adversary from performing a replay attack during the loose time synchronisation exchange between a host A and host B? Two forms of a replay attack may occur in a timestamp exchange initiated by host A with host B:

1. An adversary may have captured and stored an old response sent by host B in a previous timestamp exchange. If the adversary lies between A and B, and A initiates a loose time synchronisation with B, the adversary can replay the old timestamp response back to A instead of B's correct response. The adversary can then exploit host A's incorrect time synchronisation, by replaying old routing messages that have originated from B.

2. Host A and B perform the loose time synchronisation correctly. An adversary however can replay the timestamp request to B, thus initiating an unnecessary response from B. If the adversary continues to do this, it can waste a considerable amount of B's resources, especially considering the signature B will have to verify on each request.

To prevent such attacks from occurring, a commonly used nonce approach can be taken, as used in [52]. The loose time synchronisation exchange incorporating the use of nonces is shown in figure 3.4.

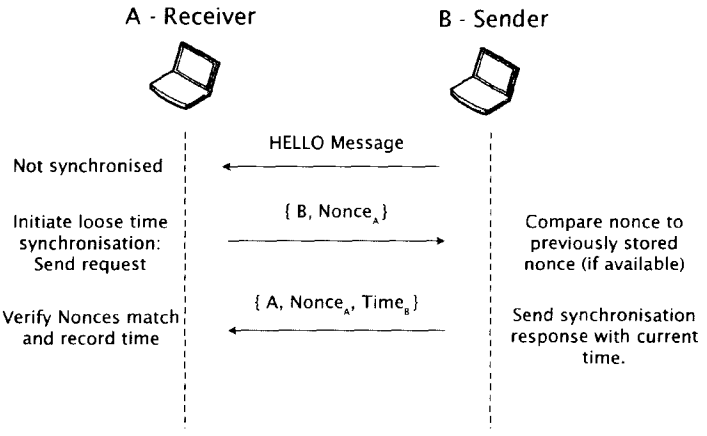



Figure 3.4: Timestamp Challenge-Response.

The time synchronisation request now contains: a nonce value based on the sender's (A) time; and the IP address of B. When B receives the request, B performs the following:

1. Verify the signature of the request. If unsuccessful, discard the request.
2. Check for a record of a nonce, stored from a previous request of A (if any). If present, compare the nonce of the previous request with the nonce presented in the current request; if the current nonce is newer, accept the request.
3. Generate a timestamp of current time, send the timestamp, A's nonce value from the request, and the IP address of A in a response message.
4. Sign the response message.

When A receives the response, it first verifies the signature of B. If verified successfully, it validates that the nonce in the response message matches the original nonce it sent in the request message. The nonce allows A to determine the freshness of the response. It ensures that no malicious node has replayed an old synchronisation response message to A; this would not be possible, as the malicious node would be unable to forge the signature of an old response message while incorporating A's current nonce, without the knowledge of B's private key. Note that the IP addresses of the parties involved are used in the request and response messages, this is due to the fact that an OLSR message is used as a carrier, and since OLSR messages are broadcast to all nodes, a means is needed to identify the parties involved in the exchange.

One can see from the timestamp verification process that the round-trip time ($\Delta$) is a very important component. Whilst the RTT calculation presents no problem between two directly connected nodes, its use may be exploited in a synchronisation exchange between two nodes several hops apart. Consider the scenario depicted in figure 3.5.

The scenario in figure 3.5 presents a non-trivial problem. If an intermediate node, such as node B, is able to deliberately delay the time a response message takes to arrive at a requester, the RTT calculated by the requester will be incorrect. By increasing the RTT that node A calculates, B has a greater window period for replaying messages that have originated from node C. If one looks at this problem in the context of the OLSR protocol, an adversary would have to deliberately hold a synchronisation response for quite a length of time; after all, the objective of an adversary is to replay routing information that reflects a state different to the current (correct) state, such that the receiver processes the incorrect information and corrupts its routing table entries.

As a possible solution, nodes could reject timestamp responses that have taken an unrealistic period of time to arrive. A node for instance, could use a threshold - based on the number of hops between it and the other party involved in the exchange - to decide whether to accept or reject a response. Determining an appropriate threshold however would be non-trivial as many factors would have to be accounted for. The mentioned solution is only a concept, and is by no means deemed to be a successful solution. Whilst this problem has been identified, and a possible solution briefly discussed, the use of such a solution has been neglected as it

was deemed unnecessary for the purposes of the experiments being undertaken in this work. Assuming a situation with synchronised nodes would negate the problems discussed above and make the timestamp verification process easier. From this discussion, it is easy to identify the additional complexities involved when using unsynchronised nodes as compared to assuming time synchronisation in an ad hoc network.

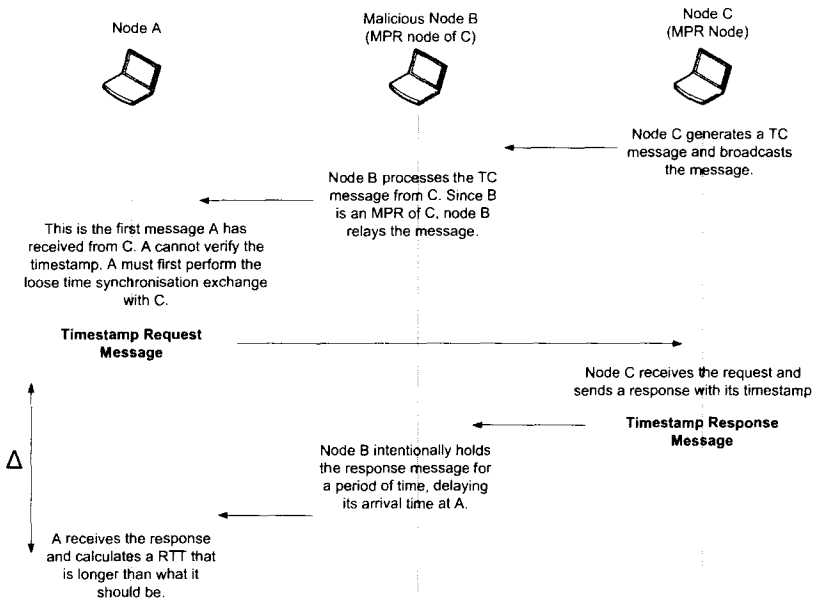



Figure 3.5: Potential problem with loose time synchronisation over multiple hops.

## 3.5 MUTABLE FIELD PROTECTION

The use of a digital signature to provide authentication and integrity of a message has been discussed briefly. However, certain fields in an OLSR routing message change from hop to hop: the Time-To-Live (TTL) field, and Hop Count field. A digital signature, computed at the time of message creation, cannot protect such fields at each hop. The following are some approaches which can be taken:

- No Protection: One approach is to simply provide no protection of the mutable fields, i.e. A sender creates an initial signature over the message, at each hop the TTL and Hop Count fields are adjusted accordingly. This approach however is open to threats. The main threat is that a malicious or compromised node can lower the Hop Count and raise the TTL fields, thus ensuring the message is transmitted over more hops than it is intended to. Additional message transmission creates additional traffic and processing overhead.

- Per-Hop Signature: At every hop, each node could create a signature over the mutable fields and append it to the message. This approach creates unnecessary overhead, both in terms of the signature size overhead and the processing overhead at each hop in creating the signatures.

- Hash Chain Mechanism: A solution presented in [68] uses two hashes to protect mutable fields from being lowered at each hop. The solution presented in [68] is used in this work, and our application of it, to the OLSR protocol, is explained below.

The hash chain mechanism involves two hash fields appended to each message: the Top-Hash and Hop-Hash fields, present in Figure 3.1. An OLSR node generating a message performs the following steps prior to sending the message:

1. Generate a random value known as the seed of size 20 bytes.
2. Set the Hop-Hash field to the value of the seed.
3. Hash the seed value TTL times, using the result as the Top-Hash field.

An intermediate OLSR node performs the following procedure when receiving a message:

1. Hash the Hop-Hash field TTL times, compare the resulting value to that of the Top-Hash field value.
2. If the two values are not equal, discard the message.
3. If the message is going to be forwarded, alter the Hop-Hash field by hashing it once.

An intermediate node is unable to modify the Top-Hash field as this field is used by the originator when computing the message's signature. The Hop-Hash field cannot be modified to reflect a lower hop-count due to the one-way property of a hash function. Hence, in the worst case, a malicious intermediate node can simply not hash the Hop-Hash field and not alter the Hop Count and TTL fields accordingly; this would simply result in the message being sent 1 extra hop.

### 3.5.1 An Extension to the HORS One-Time Signature Scheme

Providing efficient source authentication in broadcast communications is still a big challenge [51]. The HORS broadcast one-time signature scheme aims to provide an efficient authentication mechanism for broadcast communications. The HORS scheme was chosen as one of the signature schemes used in this work due to: 1) the heavy dependency OLSR has on data broadcasting; and 2) for comparative reasons, to observe the different performance characteristics between the HORS signature scheme and the IBS signature scheme based on RSA that was introduced in section 2.7.1. The use of One-time signatures in routing protocols has been adopted in other works [65, 70]. In [65], the HORS one-time signature scheme is incorporated into the AODV on-demand ad hoc routing protocol to provide efficient signatures for its routing messages. For the remainder of this section, the term public key will be used to refer to a HORS public key.

### *An Extended HORS Key Chain*

Section 2.7.1 provided a description of the standard HORS one-time signature scheme. A HORS key pair is limited to signing $r$ messages; the reason for this limitation, is due to the disclosure of the private key values ($s_i$) in each signature the private key is used for. If an adversary is able to passively listen and capture enough of these private values, it could use them to forge a signature for an illegitimate message.

In the context of the OLSR routing protocol, where messages are generated and broadcast on a regular basis, the limitation of each key pair would impose regular key pair refreshes on each node. Not only will a node have to generate key pairs regularly, but also distribute the public keys. This is evidently not optimal from a traffic overhead point of view. Extending the lifespan of a one-time key is not a novel idea, and has been presented in other works, including [65, 70]. The remainder of this section describes how the HORS signature scheme has been extended in this work.

#### Extended Key Chain Construction

The main difference to the standard scheme is in the way a key pair is created. In the standard approach, a public value $v_i$ is created as the output of a one-way function of a private value $s_i$ - which is randomly generated. The security is offered by the one-way function in the sense that only the creator knowns the value of $s_i$, and that nobody can derive $s_i$ from the public value $v_i$. Instead of hashing a private value $s_i$ just once, however, the life of a key pair can be extended by creating a hash chain of keys of length $n$ as shown in figure 3.6.

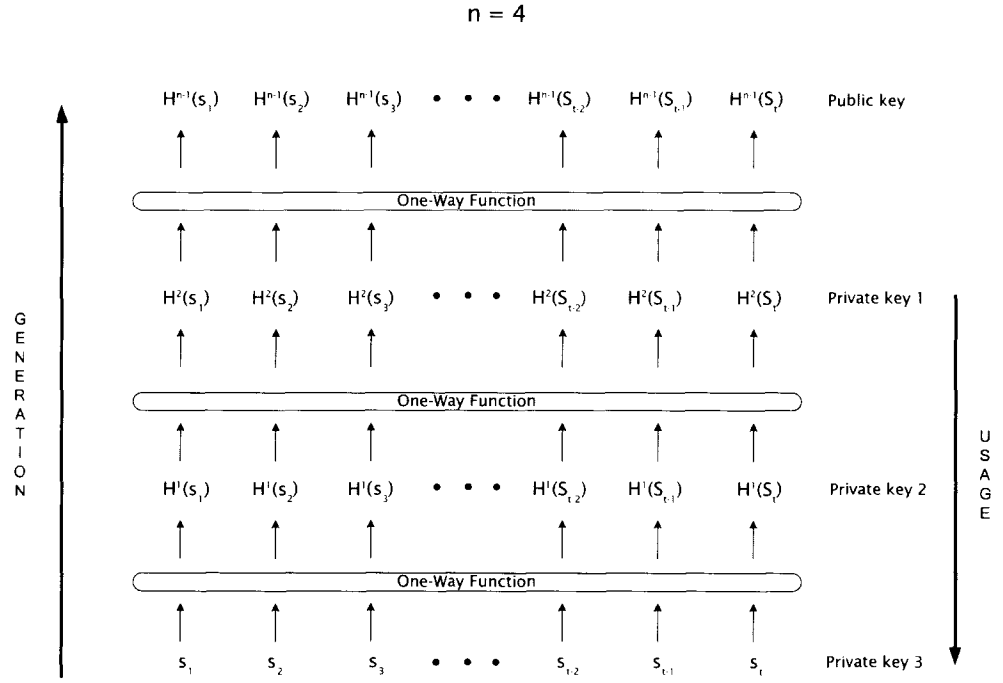



Figure 3.6: Generation of an extended HORS key pair.

Apart from the initial private key that is randomly generated, a value $s_i^j$ in a subsequent key $j$, at index $i$ of the key, is calculated as the hash of the corresponding value at index $i$ in the previous key $j-1$ as follows:

$$s_i^j = H(s_i^{j-1}) \qquad 0 < j \leq n-1$$

The extended key generation process will still produce only one public key, the difference in the extended scheme is in the number of private keys produced that correspond to the public key. The very top key (key n-1), which is the last key to be generated, still serves as the public key. The remaining keys $(0, ..., n-2)$ serve as the private keys and are used in reverse order to that in which they are generated.

**Signature Creation**

Like the standard HORS scheme, each private key from a key hash chain can be used to sign $r$ messages. Messages are signed in the same manner by disclosing $k$ values from the current private key as the signature. Once a private key $j$ has been used to sign $r$ messages, key $j-1$ is then used for the subsequent $r$ signatures. If $j = 0$, then a new key hash chain must be generated and the new public key distributed.

**Signature Verification**

In order to verify a message, a recipient needs to know: 1) the total number of keys in the senders key hash chain (including the public key); and 2) the position of the sender's private key in the sender's key chain. When verifying a signature of a message $M$ that has been signed using a private key $i$ from a key hash chain, the following high-level pseudo-algorithm (shown on the following page) is used by the recipient:

**Input**: Signature $s[k]$; Message $M$; Sender's public key $pubkey[t]$; HORS parameters $k, t, l$
**Output**: true or false

$H(M) \leftarrow$ `hash(`$M$`);`

$n \leftarrow$ Number of keys in sender's key hash-chain;

$i \leftarrow$ Position in the hash chain of the private key used to sign M;

$pubkeyindex[k] \leftarrow$ Split $H(M)$ into $k$ bit-strings of length $\log_2 t$, interpret as integers;

$hashiterations \leftarrow (n-1) - i$;

**for** $sigindex \leftarrow 0$ **to** $k-1$ **do**
  **for** $hashcount \leftarrow 0$ **to** $hashiterations$ **do**
    $s[sigindex] =$ `hash(`$s[sigindex]$`);`
  **end**
  **if** $s[sigindex] \neq pubkey[pubkeyindex[sigindex]]$ **then**
    return $false$ ;
  **end**
**end**
return $true$ ;

**Algorithm 1**: Extended HORS key Verification

The verification process is almost identical to that of the standard HORS scheme. The difference is in the number of times a signature value $s_i$ must be hashed to correctly match the corresponding value in the sender's public key. The number of hashes required is directly related to the position of the signing key in the sender's key hash chain. As more private keys from a chain are used, the time and processing requirements of the verification process will take longer. This is due to the increasing number of hash function calls required. Although this is a downside to this approach, the primary aim of the extended HORS key chain, in this work, is to decrease the traffic overhead that is otherwise imposed by key distribution requirements of the standard HORS scheme.

**Problems with the Extended HORS Approach**

There is an inherent security flaw with the extended HORS scheme mentioned above. The security flaw can be exploited when there exists an intermediate node between two communicating nodes, which is very likely in an ad hoc network. Consider the situation depicted in figure 3.7 with three nodes. A simplified example is given below to demonstrate a possible attack that exploits this security hole.

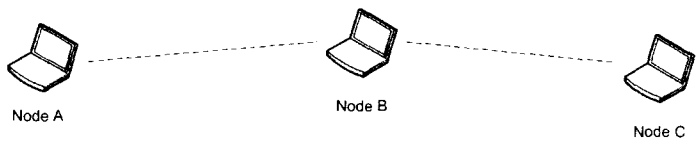



Figure 3.7: Three node topology with intermediate node B

Please note that the parameters used in this example are not realistic or practical, but rather simplified values just for the purpose of the example. The parameters used in this example are listed in table 3.1 below.

Table 3.1: Example Parameters

| Parameter | Value |
|---|---|
| No. of values in each key(t) | 8 |
| No. of signatures per key(r) | 2 |
| No. of signature values (k) | 2 |
| No. of keys in hash chain (n) | 3 |

1. Node A generates an extended key hash chain of $n$ keys$(k_0, ..., k_{n-1})$. A distributes $k_{n-1} = k_2$ as the public key for the key hash chain.

2. Node B receives and stores $k_2$. Node B then correctly forwards the public key onto node C.

3. Node C receives and stores the public key of A.

4. A generates routing messages and signs them with the first private key $k_1$ from the key hash chain.

5. B receives the messages. However instead of forwarding the messages to C, B stores the values of the private key disclosed in the signature and drops the message. B does this for all messages signed with A's first private key.

   At this point B would have received 2 messages signed with A's first private key $k_1$. Each signature discloses 2 random values from the private key. B would therefore know at most 4 unique values from A's first private key. Node B's current knowledge of A's hash chain at this point is depicted in figure 3.8. Even if node B is unable to forge a signature at this point with at most 4 unique values of A's first private key, node B is still able to obtain more values of A's first private key from subsequent private keys of A's key hash chain.

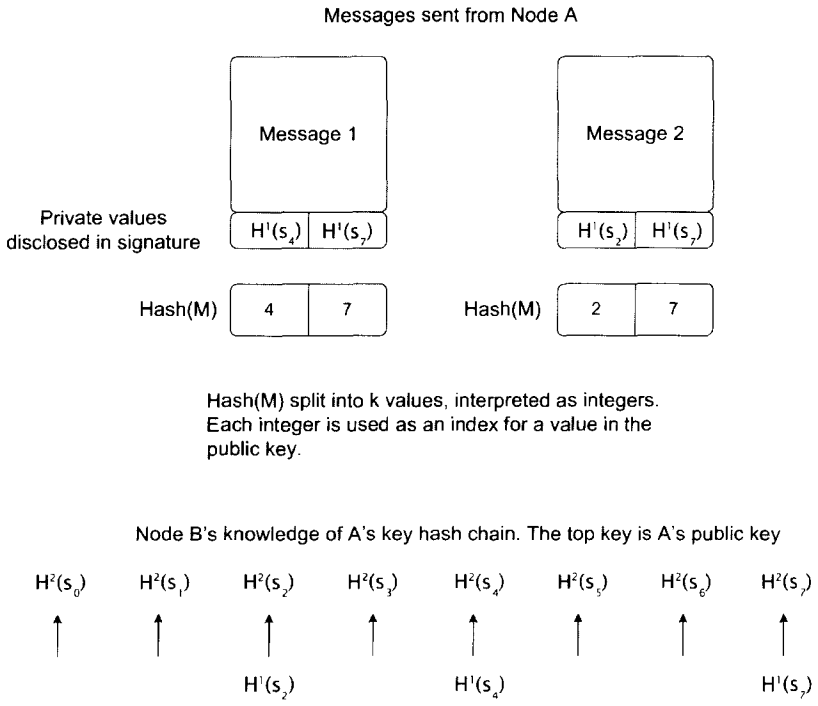
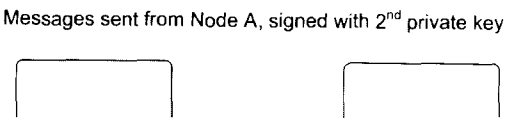



Figure 3.8: B's knowledge of A's key hash chain after A's first private key has expired

6. Node A's first private key $k_1$ has now expired after signing $r = 2$ messages. Node A now uses the second private key $k_0$ from its hash chain to sign the routing messages it generates.

7. B receives the messages. However instead of forwarding the messages to C, B stores the values of A's $k_0$ private key disclosed in the signature and drops the message. B does this for all messages signed with A's second private key.

   At this point B would have received 2 messages signed with A's second private key $k_0$. Each signature discloses 2 random values from the private key. B would therefore know at most 4 values from A's second private key. Node B's current knowledge of A's hash chain is depicted in figure 3.9.






discussed, and the pros and cons of each scheme raised. The two signature scheme primitives form part of the design process, explained in the following chapter.

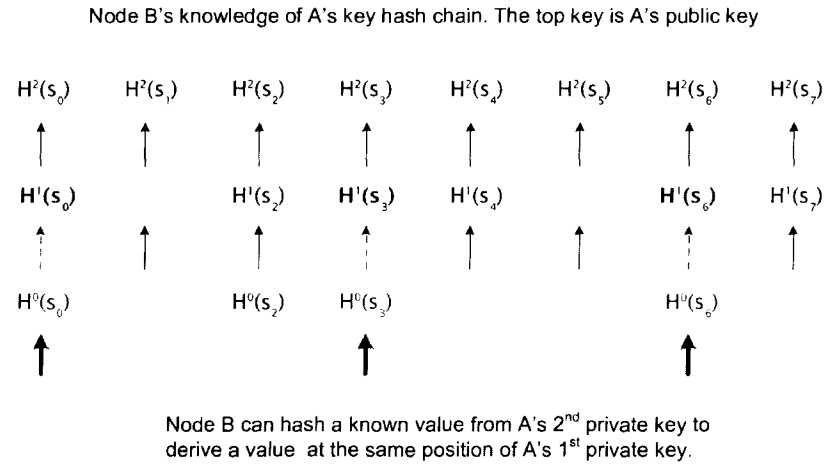



Figure 3.10: Node B deriving missing private key values

Node B now has 6 values of node A's first private key. The more values of this key node B obtains, the higher the probability that node B can successfully forge a signature of node A. As more private keys are used by A for a given key hash chain, the chances of B deriving more values - of an expired private key of A - increase.

8. Node B may now be able to generate an illegitimate routing message, forge A's signature for it, and send it to node C. Node C receives the message, verifies the signature against the public key of A which it had previously received. On successful verification, C processes the incorrect routing information of the message.

As already stated, this example made use of unrealistic parameters. In a practical situation, realistic parameters would provide better security. However, the principle behind the security threat still applies and the problem must be addressed. The solution used in this work enforces a validity time on each private key in a key hash chain. The use of such a solution, in the context of the OLSR routing protocol, is explained further on in section 3.7.2.

**Extended HORS Key Resource Requirements**

Whilst the extended key scheme provides a means to sign more messages per public key, the scheme does however impose greater overhead in terms of signature size. Owing to the fact that values in a private key are constructed as hash output of a previous private key in a key chain, each value in a private key (l parameter) is 160-bits in size as opposed to 80-bits in the standard HORS scheme. In essence, signatures created using the extended HORS scheme are double the size of those created using the standard HORS scheme. The public key size however remains the same in the extended HORS scheme.

The extended HORS scheme however has a significant benefit in the fact that a public key and its corresponding private keys remain valid for longer periods of time. This means that public key distribution takes place less often, resulting in less traffic overhead considering the large size of public keys. Figure 3.11 shows the overhead (in bytes) per signature provided, depending on the number of private keys in an extended HORS key chain.

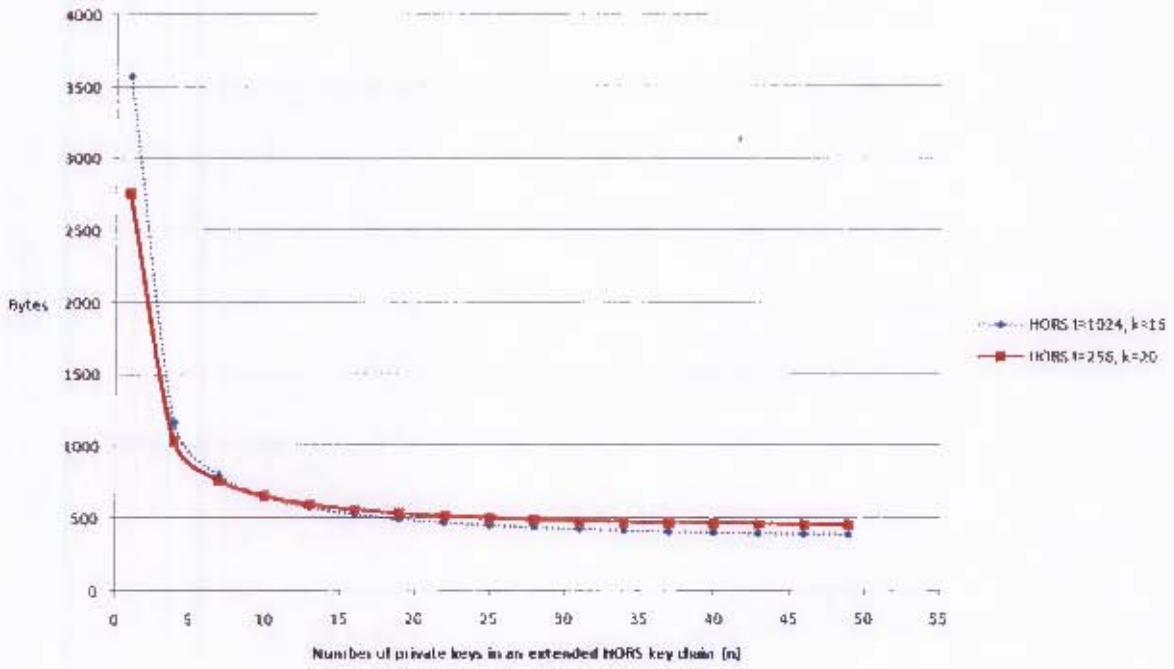



Figure 3.11: Overhead (bytes) per signature provided

The graph represented in figure 3.11 has been plotted using the following calculation:

$$(public\_key\_size + (signature\_size \cdot signatures\_per\_key\_chain))/signatures\_per\_key\_chain$$

$$- (t \cdot hash(l) + ((k \cdot l) \cdot (r \cdot n)))/(r \cdot n)$$

Where $(t, k, l, r)$ are parameters to the HORS scheme, and $n$ is the number of private keys in an extended HORS key chain. If $n = 1$, the scheme is essentially the standard HORS scheme with a single public and private key, and with each value $l$ being 80-bits in size. Once $n > 1$, the scheme represents the extended HORS scheme. Even though signature sizes are greater in the extended HORS scheme, one can see from figure 3.11 that the benefit of less frequent key refreshes becomes far greater as $n$ increases. Whilst the traffic overhead decreases as $n$ increases, it must be noted that the storage size of an extended HORS key chain increases with $n$. This fact must also be considered when determining the correct value for $n$, especially if storage size is limited, as is the case with many embedded systems. The value of $r$ above, for both parameter sets, was chosen such that both parameter sets offered roughly the same strength of security (in bits): for HORS 1024, $r = 6$; and for HORS 256, $r = 2$.

Although differences exist between the standard HORS scheme and the extended HORS scheme introduced in this section, for simplicity sake, the terms *HORS* and *extended HORS* are used

interchangeably to refer to the extended key chain version of HORS. This is the case for the remainder of this dissertation. If the original, standard HORS scheme is referred to, it is explicitly stated so.

## 3.6 CONFIDENTIALITY OF OLSR PACKETS

Optional encryption is provided for every OLSR packet on a hop-by-hop basis. Encryption prevents outsiders from passively eavesdropping on the contents of the routing messages. The OLSR packet will require additional headers to allow the receiver to determine whether or not encryption was used on the packet's contents. The revised OLSR packet with security headers is shown in figure 3.12.

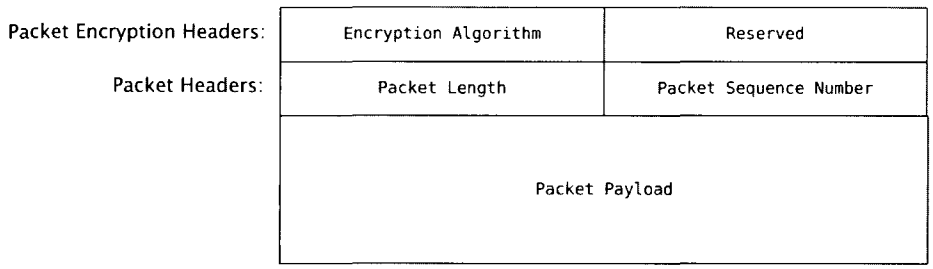



Figure 3.12: OLSR Packet with encryption header

Both the security headers and original OLSR packet headers are left unencrypted to allow the receiver to interpret them; only the messages in the packet payload are encrypted. The Encryption Algorithm field specifies which encryption algorithm was used to encrypt the content of the packet payload. Currently only two values have been set aside for this field: 0 - no encryption; and 1 - AES encryption. The reserved field is present for any additional security parameters that may be needed, and also to keep the headers 32-bit word aligned.

## 3.7 TWO METHODS TO SECURING OLSR

Prior to this, various security mechanisms that will be used to secure OLSR have been explained. Two separate asymmetric signature schemes have been discussed, including an extension to the HORS scheme. This section describes how two different signature schemes are incorporated into this OLSR security design, resulting in two different methods. One method utilises the IBS signature scheme explained in 2.7.1; whereas the other method is a hybrid approach that utilises both the IBS signature scheme and the extended HORS signature scheme. The use of two different methods allows one to compare the differences at a performance and overhead level between the two approaches.

### 3.7.1 The OLSR IBS Signature Scheme Method

For this method, the IBS scheme constructed by Shamir, explained in section 2.7.1, is used to provide signature generation and verification operations; for simplicity sake, this signature scheme is referred to as the IBS signature scheme for the remainder of this thesis. The main advantage of using the IBS scheme is that there is no need to distribute public keys. This feature is useful in a dynamic environment, such as an ad-hoc network. Consider the case of using a

traditional digital signature scheme, such as RSA, to sign OLSR control messages. When a node digitally signs a message with its private key and broadcasts the message, all recipients will have to be in possession of the sender's public key to verify the message. Three possible approaches to achieving this are described below:

1. Pre-shared Public Keys: This approach requires the public keys of every node to be pre-shared on all nodes. In a network of N nodes, each node would thus have to store N public keys (including its own). This approach is also not very scalable; a new node cannot join the network without having its public key pre-shared on every other node.

2. Broadcast Public Keys: A node could broadcast its public certificate, periodically, or by request from another node. A node's certificate would have to be signed by a trusted authority and bootstrapped on the node prior to entering the network. This approach is more scalable than the previous approach, however produces more traffic overhead in the network.

3. Trusted Central Key Authority: This approach is the most likely approach in a traditional centralised network. However, in an ad hoc environment, the presence of a central key server cannot be assumed.

Consider the same scenario, instead using the IBS scheme to sign OLSR control messages. When a node digitally signs a message with its private key and broadcasts the message, recipients can calculate the sender's public key using publicly known information about the sender, such as its IP address for instance. Using such a scheme prevents the need to distribute public keys amongst communicating entities, which is often a non-trivial task in ad hoc networks. The IBS scheme provides better scalability, particularly in the case where keys are pre-shared. Using the IBS scheme, a node only requires its private key along with the master public key to be pre-shared. This is compared to N public keys, in a traditional public key system with N nodes.

***Signing OLSR Messages***

Under the OLSR IBS signature scheme method, the signature creation process of a routing control message is straightforward. To sign an OLSR message, an IBS signature is created as the signature field shown in figure 3.2.1. The data fields represented by a signature are those already mentioned in section 3.3.

***Initiating Loose Time Synchronisation under the IBS Signature Approach***

In section 3.4.3, it was stated that if a node receives an OLSR message from another node it has no time synchronisation with, the recipient node must initiate a loose time synchronisation exchange; however, when processing incoming OLSR messages, this is not always the case. A recipient node must only initiate a time synchronisation exchange if the received OLSR message is in fact a routing control message (HELLO, TC, HNA, MID), and not a security-related message, such as a time synchronisation request. The reason behind this being that routing control traffic may only move in one direction between a pair of nodes more than a single hop apart. For instance: An MPR node will broadcast a TC message that will traverse the entire

network. The MPR node may at a certain time receive a time synchronisation request from a non-MPR node that is not a direct neighbour. The MPR node will respond to the request, however, due to the fact that the MPR node may not receive any routing control messages from the non-MPR node, there is no need for the MPR node to be loosely synchronised with the non-MPR node. This approach reduces the complexity and overhead involved with the time synchronisation exchange process.

### *Processing OLSR Packets*

From a security point of view, the most important operations are: securing the outgoing OLSR packets; and processing incoming security-aware OLSR packets. Figure 3.13 shows a high-level overview of the procedures and decisions involved when processing incoming OLSR packets under the IBS signature scheme approach, whilst figure 3.14 represents outgoing OLSR packets.

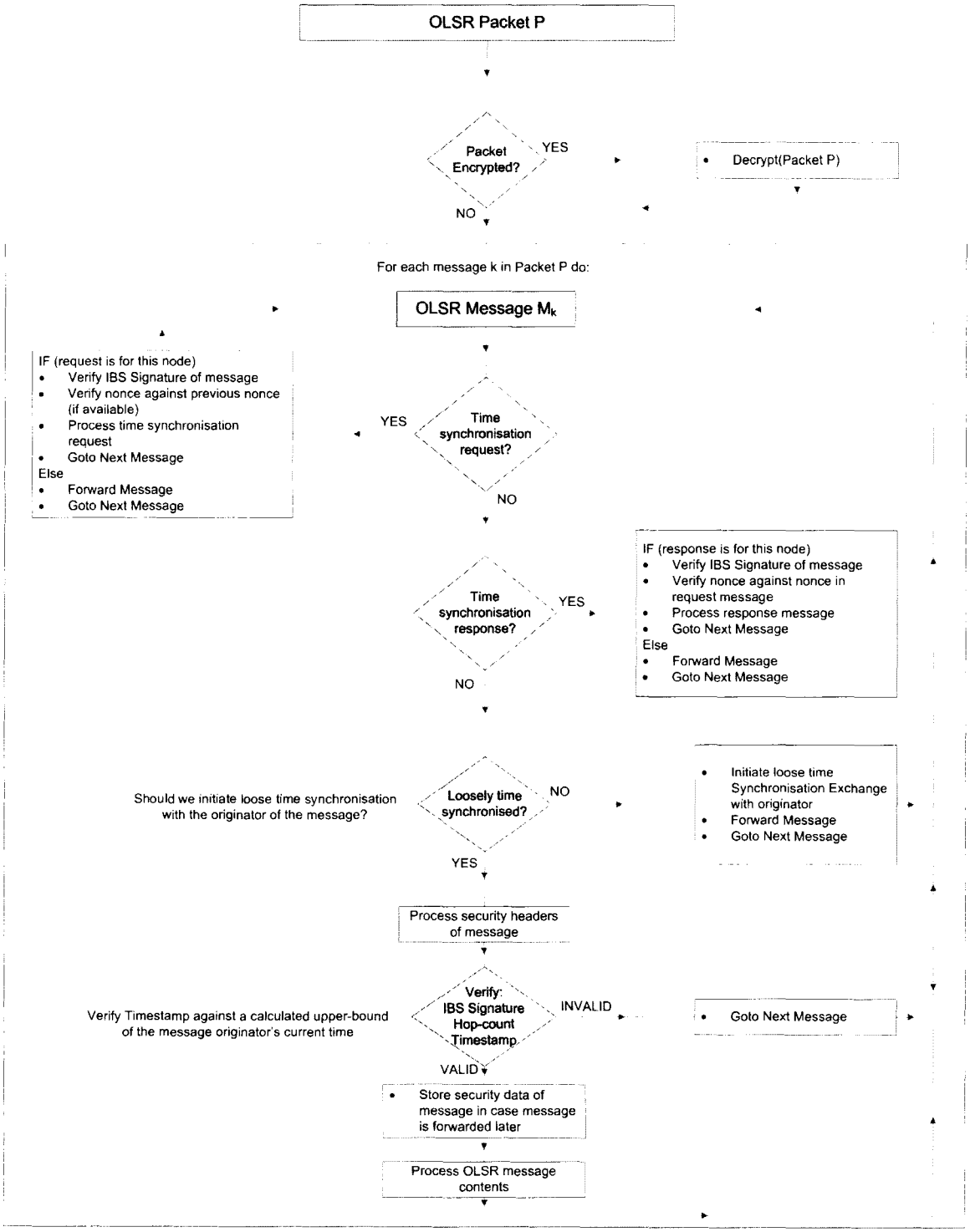



Figure 3.13: Processing incoming security-aware OLSR packets

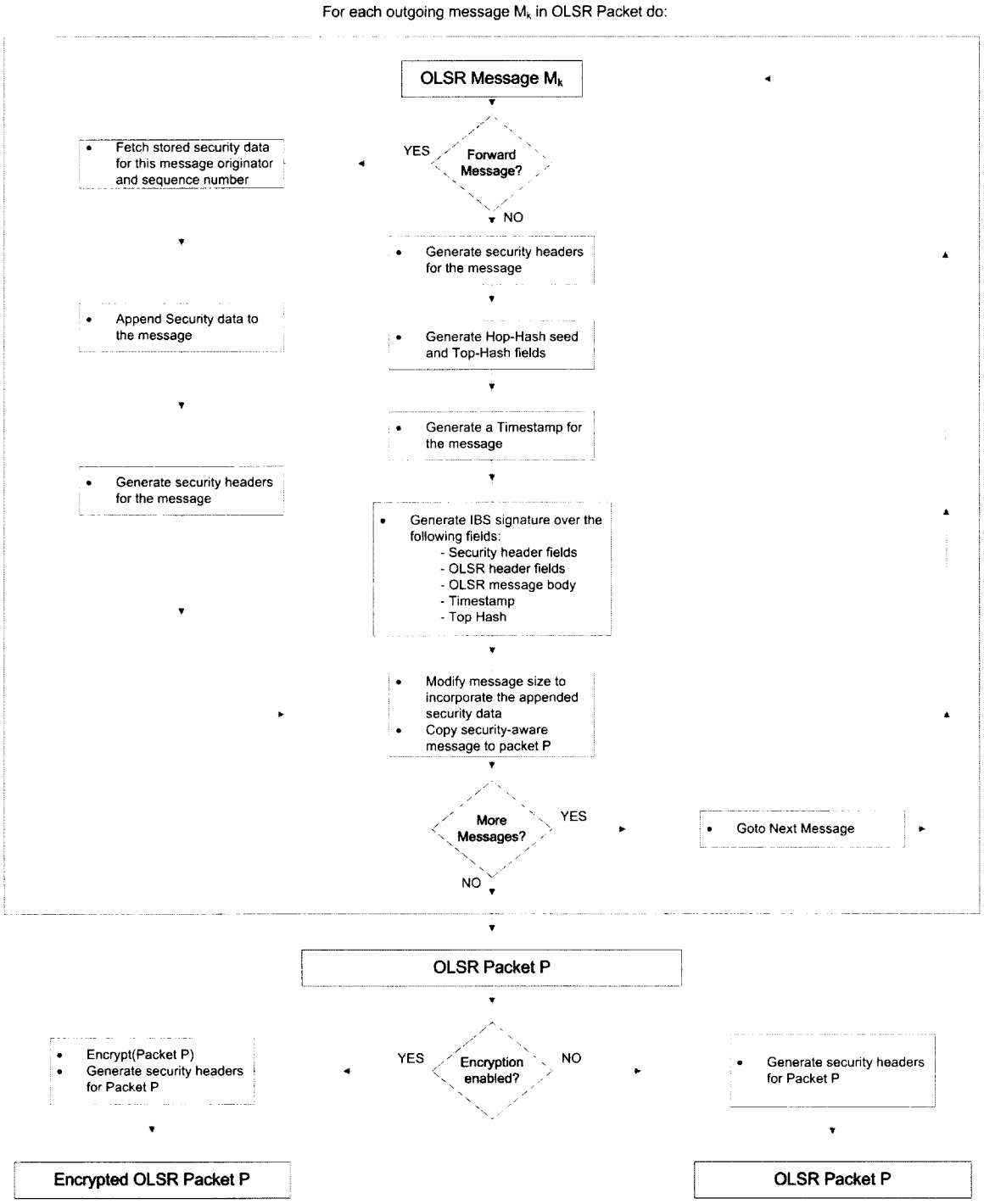



Figure 3.14: Processing outgoing OLSR packets

### 3.7.2 The OLSR Hybrid Signature Scheme Method

The hybrid signature scheme method requires the use of two different asymmetric signature schemes. The primary signature scheme used for this hybrid approach is the extended HORS one-time signature scheme, discussed in section 3.5.1. For the remainder of this section, the terms public key and private key refer to an extended HORS public key and private key respectively. References to public and private keys of other signature schemes are made explicitly.

#### *Signing OLSR Messages*

Unlike the OLSR IBS signature scheme method explained in section 3.7.1, the process of generating a signature for an OLSR message is dependent on certain factors. For a given OLSR message, these factors determine how the message will be secured: 1) using the general security-aware format shown in figure 3.2.1; or 2) whether the security of the message be handled unconventionally, as a special case. The remainder of section 3.7.2 explains the factors and conditions involved with securing a given OLSR message under the hybrid signature scheme method.

Owing to the fact that the HORS signature scheme is used as a primary signature scheme, we first show the HORS signature format when used to secure a security-aware OLSR message. The format is shown in figure 3.15.

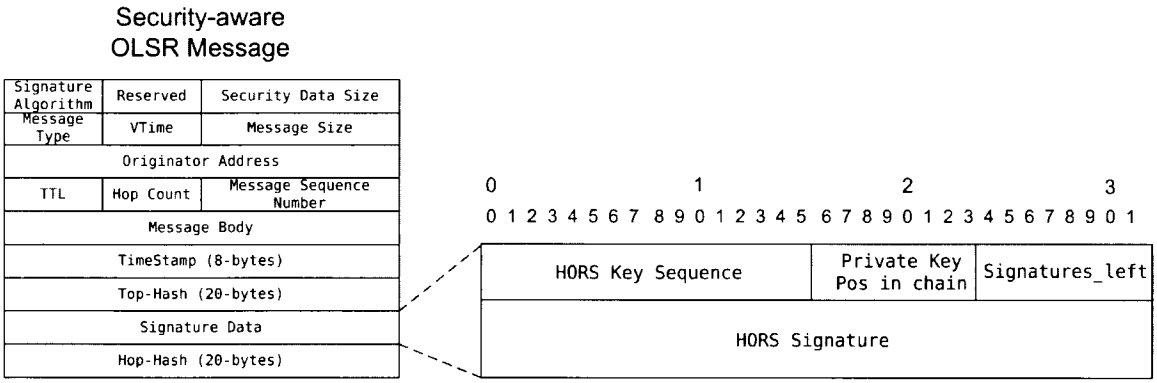



Figure 3.15: HORS signature of a security-aware OLSR message

A node must assign its key hash chain an identifier; each time a node generates a new key hash chain, the identifier is incremented and assigned to the new chain. The *HORS Key Sequence* field allows a recipient node of a message to identify which key chain was used by the sender to sign the message, and if the recipient is in possession of the correct public key of that key chain. The *Private Key Position in chain* field allows a recipient node to determine the number of hashes required to verify each value disclosed in the signature against its corresponding value in the sender's public key. The *Signatures Left* indicates how many signatures may still be produced by the current private key of the message originator. This field is used in the calculation of a validity time-frame for a HORS private key, discussed later in section 3.7.2.

### *HORS Public Key Distribution*

When a HORS key chain is generated, neighbouring nodes will require the public key in order to verify any received messages that have been signed with a corresponding private key. A mechanism is therefore required for the distribution of a HORS public key between nodes. Two potential problems with this scenario exist:

1. To facilitate a public key exchange between two nodes - more than one hop apart - it is best that the nodes in question possess routes to one another, and that the intermediate nodes know how to forward the data between the two end nodes. However, in the context of security and the OLSR protocol, routes cannot be discovered until the routing messages have been processed, which in itself, relies on the signatures of the routing messages being verified correctly. Hence, bringing us back to the key exchange process.

2. The mechanisms of the HORS signature scheme are very different to those of regular asymmetric public key algorithms such as RSA. An RSA key for example, has a longer lifespan than that of a HORS key, which is bound to a limited number of signatures. This brings about the problem of secure key distribution of a HORS public key. For instance: RSA public keys can be securely pre-shared/bootstrapped on nodes prior to a network going live. This is valid due to the fact that the public keys and their corresponding private keys will remain valid for a relatively long period of time. A HORS public key on the other hand, remains valid for a very short period of time, making a pre-shared/bootstrap option of a public key invalid and impractical. Hence, a mechanism to securely distribute a public key, at any time during the live operation of an OLSR node, is required.

A solution to the first problem is to use OLSR messages and the OLSR MPR mechanism as a carrier for the public key. This essentially means that 1) no routes are required prior to distributing a key, and 2) the OLSR key messages will be forwarded correctly due to the OLSR default forwarding mechanism. The downside is that whenever a key is distributed, it is always broadcast; this is not such a bad side-effect however, as in some cases more than one node may require the new public key being distributed. The custom OLSR message format used to carry a HORS public key is explained in further detail in section 3.7.2.

Now, if OLSR messages are used as a public key carrier mechanism, when, and under what circumstances should a public key distribution occur? Lets consider the point of view from two entities: a node A, generating a HORS key chain and distributing the public key; and a node B, joining the network, however, after A has distributed its public key. From A's point of view, it must distribute its public keys immediately after generating a new key chain - this is obvious. Node B however, has two options: the one option simply relies on waiting for the next key refresh of A; whereas, the other approach utilises a key request/response mechanism.

**Approach 1: Wait for Key Refresh**

Node B has joined the ad hoc network, however, after A's public key distribution. Consider the case of the standard HORS signature scheme where a public/private key pair consists of a single public key and a single private key. Owing to the fact that a HORS key pair has such a short lifespan, node B could simply wait until A's next key refresh. This approach may be acceptable under the conditions of the standard HORS scheme and a small $r$ parameter. However, under the extended HORS key scheme mentioned in 3.5.1, it may be the case where the time interval, between public key distributions of new key chains, is too long for this approach to be acceptable.

**Approach 2: Key Request/Response**

A more suitable approach is to implement a HORS key request/response mechanism. For a recipient node of an OLSR message, the following conditions must be met prior to sending a public key request:

1. The message is signed with a HORS private key of the message originator.
2. The recipient is not in possession of the corresponding HORS public key of the sender.
3. A time threshold has passed since the last key request (if any) sent to the originator of the message.

Upon receiving a key request, the recipient performs the following checks prior to distributing its public key:

1. A time threshold has passed since the last HORS public key distribution (if any). This time threshold is in place to prevent excessive key distributions taking place within a short time period. For instance: if two key requests are received by a node, from two different sources, within a short time-span, this threshold prevents two public key distributions taking place - which would be unnecessary.
2. The signature of the key request verifies, and, if the source of the request has sent a key request previously, the nonce of the current request is more recent.

**A Signature for the HORS Public Key Distribution**

Going back to the second problem listed at the beginning of section 3.7.2; to solve this, a second signature scheme has to be used to sign a HORS public key prior to distributing it. Hence, the reason it is a hybrid signature scheme method. The main requirement of the secondary signature scheme is that its keys have a long lifespan relative to that of a HORS public key. This affords the option of pre-sharing the necessary keys, of the secondary signature scheme, on the nodes prior to going live. These keys will remain valid for a lengthly period of time, and hence, can be used as a basis for secure distribution of a HORS public key. There exist several valid

signature schemes that could be used for this purpose, however due to the fact that the IBS signature scheme is already being used in this work, we have chosen the IBS signature scheme as a secondary signature scheme.

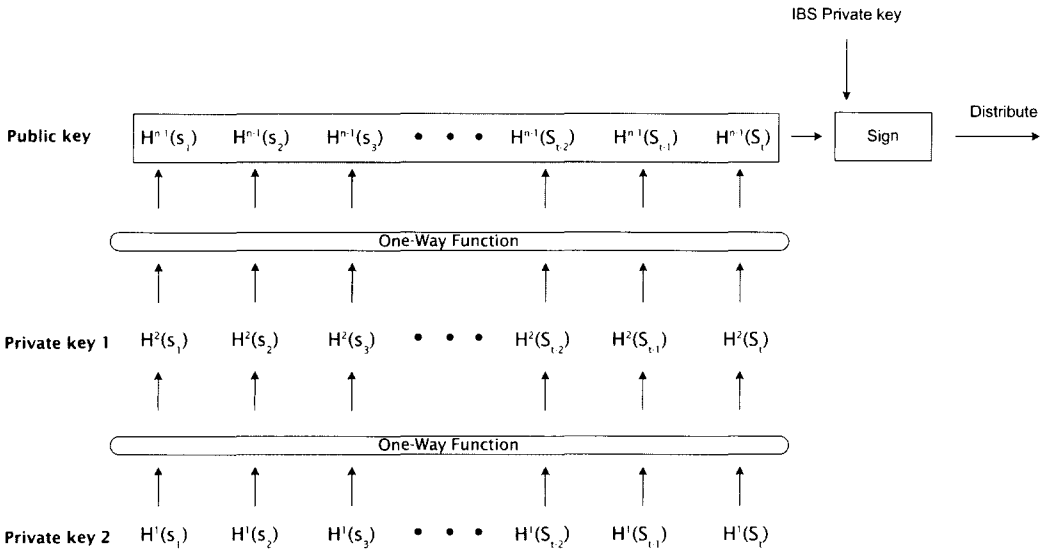



Figure 3.16: Signature for a solicited key distribution

A simple approach to using a secondary signature scheme would be to sign a new public key prior to distributing it. Therefore a single key chain would require one IBS signature, and in turn, provide $r$ signatures for every private key in the key chain before the key chain becomes invalid. It is already easy to see that a trade-off exists: the lower the parameter $r$, and the smaller the size of a key chain, the larger the dependency on the secondary IBS signature scheme. A large dependency on a secondary signature scheme is not ideal; the reason, being that HORS was chosen as the primary signature scheme due to its efficient signature creation and verification. Therefore, the aim is to minimise this dependency on a secondary signature scheme. A better approach is to use a secondary signature scheme only to sign a public key in response to an solicited public key request, shown in figure 3.16.

On the other hand, when an unsolicited public key distribution takes place, such as when a node generates a new key chain, the new HORS public key is signed using the previous private key, from the previous key chain. This approach is depicted in figure 3.17, and minimises the dependency on the secondary signature scheme. Take note that when a public key is signed using a HORS private key from the previous key chain, the signature is always the $r^{th}$ signature of the last private key, from the previous key chain.

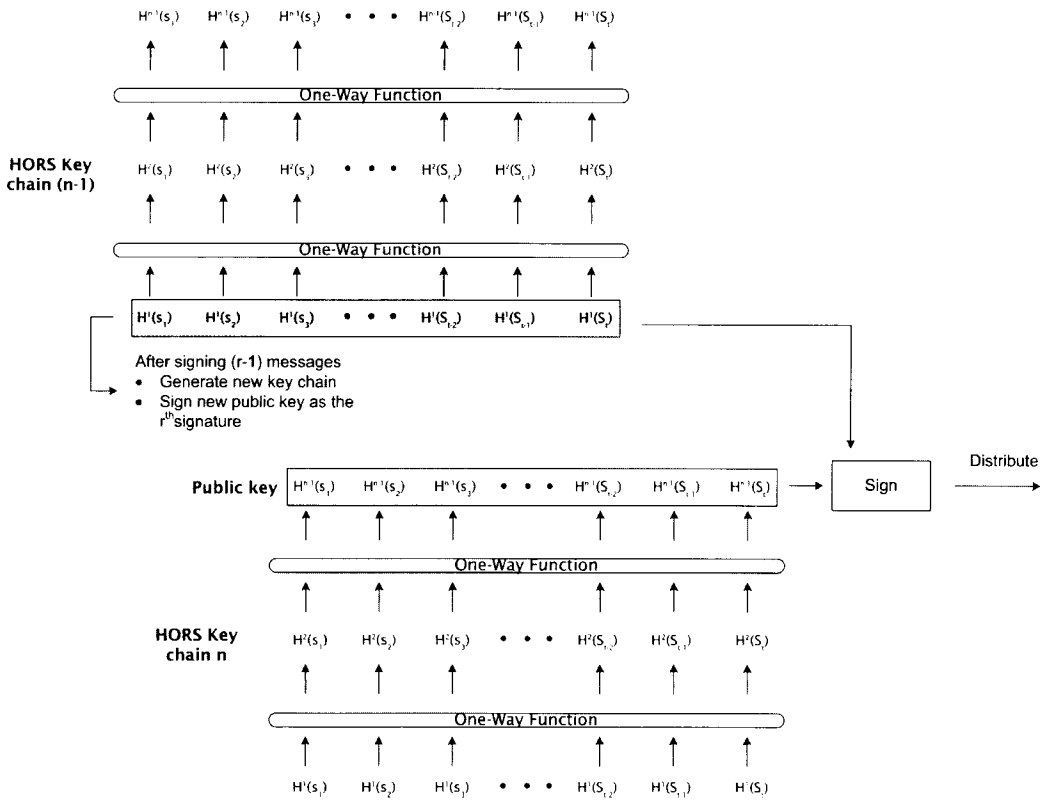



Figure 3.17: Signature for an unsolicited key distribution

## Custom OLSR Messages for HORS Public Key Distribution

It has already been mentioned that an OLSR message will be used as a carrier mechanism, for a HORS public key, for the purposes of key distribution. This section is concerned with the format of the custom HORS public key distribution message. The format of the message is shown in figure 3.18; like all OLSR custom messages, the distribution message will be carried as the body of a generic OLSR message, however, with security headers. The process of securing the public key carrier messages is handled in a special manner, this process is explained later in this section.

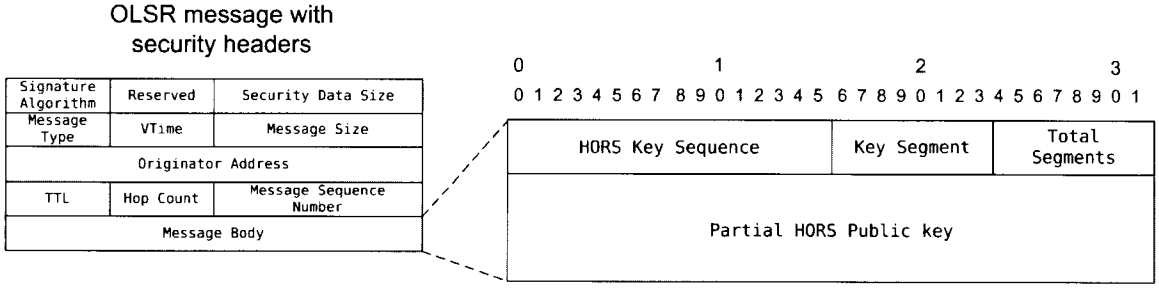



Figure 3.18: Format of custom OLSR public key distribution message

Owing to the fact that a HORS public key is quite large in size, as discussed in 2.7.1, a public key will have to be split-up and carried over several OLSR messages. The fields of the public key message are explained below:

- HORS Key Sequence: The HORS key sequence represents an identifier for the key chain that the HORS public key belongs to, this is required as key chains are refreshed on a regular basis.
- Key Segment: A public key is transmitted over several messages due to the size of the key. This identifies which segment of the overall public key this message is carrying.
- Total Segments: This fields represents the total number of messages required to transmit the public key. This allows a receiving node to determine when all the necessary partial key messages have been received.

Now the question must be asked: how does one secure each of these public key messages? A HORS public key with parameters: $l = 80$; $t = 1024$; and $k = 16$, would be of size 20480 bytes, as pointed out in section 2.7.1. Consider an OLSR message size, constraint by the Maximum Transmittable Unit (MTU) - in this case 1500 bytes. Take into account the size of various headers: IP (20 bytes); UDP (8 bytes); OLSR packet with security headers (8 bytes); and OLSR message with security headers (16 bytes). This leaves a maximum of 1448 bytes that may be carried as the message body of an OLSR message. Therefore, to distribute a HORS public key using these parameters, approximately 15 OLSR messages of maximum size would be required, each transmitted within a different OLSR packet.

It would be infeasible to sign and append security data to each of the 15 OLSR messages used to distribute a public key, as this would in most cases require more signatures, than the public key being distributed, could provide. To get around this, the security of the distribution messages is handled differently: the entire public key (20480 bytes in this case) is used in the process of generating a single signature, the public key is then split-up and sent using different OLSR messages, none of which are secured individually. The signature, as well as a timestamp, is then sent as an additional custom message, called the distribution signature message, shown in figure 3.19. Whilst each public key distribution message is not secured individually, a receiver is still able to verify the authenticity, integrity, and timestamp of a distributed public key once all the partial key messages have been received. This approach is more suitable as it only requires 1 signature for the public key distribution process. The signature is created over the entire public key, a timestamp, and the various headers shown in figure 3.19.

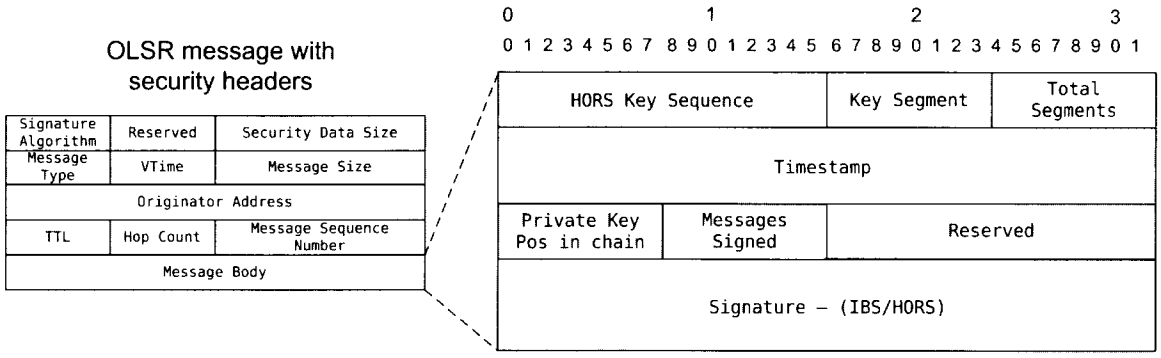



Figure 3.19: OLSR Signature message for HORS Public key distribution

Take note of the following two fields shown in figure 3.19:

- Private Key Position in Chain: A node distributing its public key, uses this field to disclose the position of the current HORS private key in its key chain. With this knowledge in hand, a recipient node knows only to accept a signature created by a private key no older - in the key chain - than that specified in this field.
- Messages Signed: This field discloses the number of signatures already produced by the originator's current HORS private key. This value is required by a recipient in the calculation of the remaining validity time of the originator's current private key. The time validity of a HORS private key is explained later in section 3.7.2.

This information is most important in the case of a solicited public key distribution. Owing to the fact that a solicited public key distribution may occur at any time, it may be the case that the node distributing its public key has already used a subset of the private keys from its key chain. Hence, recipient nodes of the public key being distributed, require the information listed above for verification of subsequent signatures created using the same key chain.

In a HORS public key distribution, it is highly possible that a recipient does not receive all the public key packets successfully from the source node. If so, the recipient node will have to send another HORS public key request to the source in an attempt to obtain any remaining public key packets it has yet to receive. This subsequent request may lead the source node to distribute its entire HORS public key once again. One could ask the question: why doesn't the recipient node send a specific request to the source node, requesting only the public key packets it has yet to receive? Whilst this is a viable option, it has been decided against in this work for the following two reasons:

1. In a wireless multi-hop environment, it is highly possible that a subset of HORS public key packets - representing the same public key from a source node - are not received successfully by one of more recipients, in a public key distribution. Instead of the source node having to service several specific requests, for different subsets of these public key packets, a single generic response of an entire HORS public key distribution can attempt

to provide all recipients with their missing HORS public key packets. As mentioned previously in this section, a recipient node of HORS public key requests uses a time threshold window to minimise excessive HORS public key distributions.

2. If such specific public key requests were used, any single destination node of such requests could end up being overwhelmed with such requests and their subsequent replies. For instance: each of these requests would require a specific reply, which itself would require unique security data (digital signature, timestamp, etc.). Instead, using a single signature for the general distribution of an entire HORS public key - which every recipient node can utilise - is the approach we have taken.

To reduce the traffic and processing overhead involved in the public key distribution process, the TTL field of every message involved in the public key distribution process is set according to the role of the node distributing the public key. Special nodes such as: MPR, HNA, or MID nodes, must set the TTL value to a maximum pre-defined TTL value. On the other hand, regular nodes must set the TTL value of these messages to 1, as the only routing control messages these nodes produce are 1-hop HELLO messages.

***Initiating Loose Time Synchronisation***

As mentioned in section 3.7.1, there are certain conditions under which a recipient node must initiate a loose time synchronisation exchange. The loose time synchronisation is not always required in both directions between a pair of nodes. The exchange must only be initiated if a message being received falls under one of the following types:

- OLSR routing control message (HELLO, TC, HNA, MID)
- HORS Public key distribution message

***Request/Response Messages***

Time synchronisation and key request messages, as well as time synchronisation response messages are treated as a special case under the hybrid signature scheme method. These message types must always been secured using the general security-aware format of figures 3.1, however the signature must be created using a secondary signature scheme - in our case, the IBS signature scheme. From a time synchronisation point of view, this is quite clear, as loose time synchronisation always occurs prior to a HORS key exchange. Hence, the need to use a second signature scheme for the time synchronisation exchange.

From a key request point of view, if a node B sends a public key request to a node A, however, signing the request with a HORS signature, there is no guarantee that node A is in possession of the correct public key of node B. A secondary signature scheme therefore makes the process of time synchronisation and key exchange less complicated, and provides security in the initial phase prior to a HORS public key exchange.

### *Time Validity of an Extended HORS Private Key*

In section 3.5.1, we pointed out a potential problem with the extended HORS signature scheme. The problem allows expired private keys to be used to derive values in subsequent private keys belonging in the same key chain. In [70], an attack of this nature is described as a Delay-and-Forge attack. As a possible solution, a validity time window is enforced on each private key in a key hash chain. The use of time intervals and windows for one-time key hash chains is discussed in [70]. This section explains how we enforce a time validity window on a private key from an extended HORS key chain.

The interval time of OLSR HELLO messages is used, as a basis, in the calculation of the time threshold for each private key. To demonstrate this, we use the message interval values shown in table 3.2 to demonstrate how the validity time for each private key is calculated. This demonstration considers an MPR node which is required to broadcast both HELLO and Topology Control messages at the intervals specified in the table below.

Table 3.2: Example Parameters

| Parameter | Value |
|---|---|
| Hello Message Interval | 2 |
| Topology Control(TC) Interval | 5 |
| HORS Parameter r | 3 |

Generally, and more specifically in the default case, an OLSR node generates HELLO messages at a more frequent interval relative to other message types. An upper-bound validity time for each private key, in a HORS key hash chain, can therefore be calculated based on the interval of HELLO messages $HELLO_{int}$, and the parameter $r$ of the HORS scheme - specifying the maximum number of messages any one private key may sign. Consider the scenario depicted in figure 3.20: Node A secures routing control messages using the HORS signature scheme. Upon receiving a public key distribution from node A, recipient nodes (node B in this case) must calculate a validity time threshold for node A's current private key. This time threshold can be determined using the following calculation:

$$T^{0}_{key\_valid\_time} = (HELLO_{interval} + slack) \cdot (r - messages\_signed)$$

Where the value $messages\_signed$ is disclosed during a public key distribution, as shown in figure 3.19. This calculates an allowed upper-bound time for which the remaining signatures, of the sender's (node A) current private key, can be accepted. The calculation however, may in some cases, be too strict. Hence, some slack is introduced to allow some difference between the expected arrival time-frame of another node's signed messages and the actual time-frame of those messages. Referring back to figure 3.20, node A initially performs an unsolicited public key distribution; because the distribution is unsolicited, it means that A has just generated a new key chain, and hence, has yet to use a HORS private key for signature purposes. Therefore, if we use the values specified in table 3.2, $messages\_signed = 0$, and $slack = 0.25$, we can calculate the validity time of node A's first private key to be 6.75 seconds.

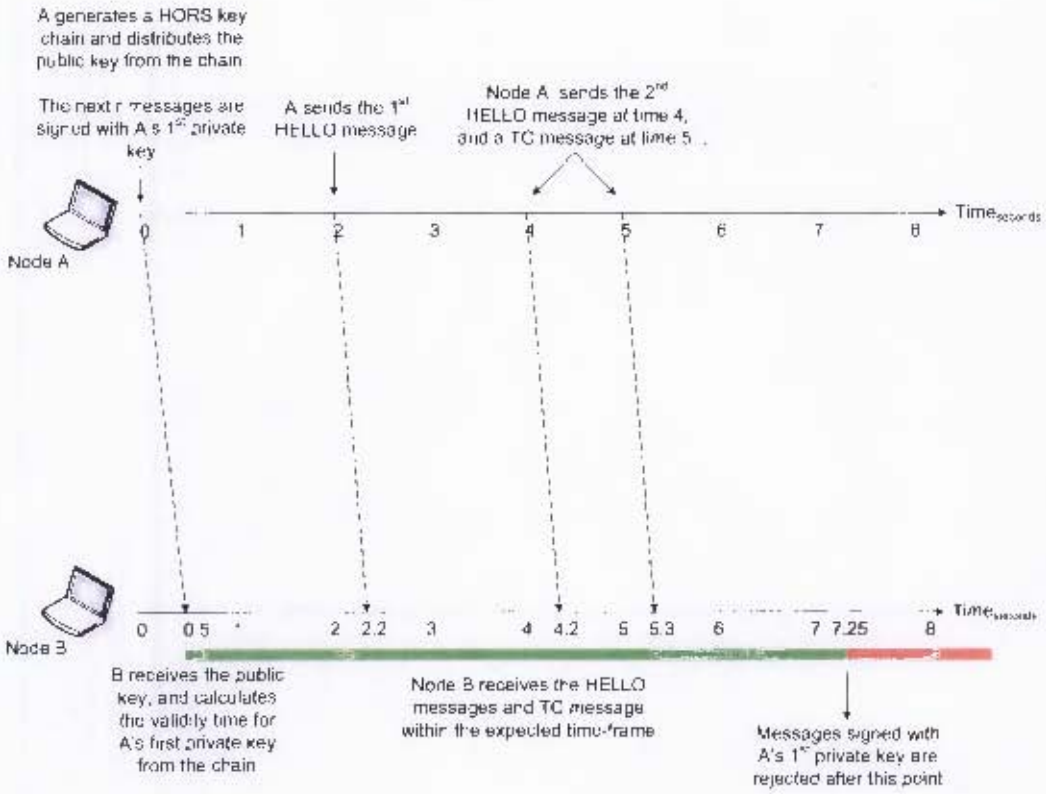



Figure 3.20: Time validity of an extended HORS private key

The validity time calculated is then added to the recipient's (node B) current time, resulting in a future cut-off point in time after which no more messages, signed with the private key in question, will be accepted by node B. Now, if node A signs messages with its second private key, node B will need to calculate a new time-frame for the private key. One simple approach could be to simply calculate a new time window and add it to the cut-off time of the previous private key to obtain a new cut-off point in time in the future. However, one can see from figure 3.20, that it may be the case that there is a significant amount of slack between the calculated time-frame and the actual time-frame of the incoming messages. The impact of this slack needs to be minimised prior to calculating a time-frame for node A's next private key. To achieve this, a record of two things should kept whenever a message is received successfully: 1) the number of signatures that can still be produced, by the current private key of the message originator. This value is indicated as part of a HORS signature, as shown in figure 3.15; and 2) the current time. These two values are used when calculating a valid time-frame, for a subsequent private key, as an extension to a time-frame of a previous private key belonging in the same key hash-chain.

Returning to the example above, assume node B does not receive the 3rd message from node A. The next message B will receive from node A will be signed using a subsequent private key of node A. For node B to calculate a time-frame for the subsequent private key, belonging to the same key hash chain as the previous private key, B can use the following calculations:

$$T^0_{cut\_off} = last\_msg\_timestamp + ((HELLO_{interval} + slack) \cdot signatures\_left)$$

$$T^1_{key\_valid\_time} = T^0_{cut\_off} + ((HELLO_{interval} + slack) \cdot r)$$

The first calculation attempts to mitigate any excess slack of a valid time-frame calculated for a previous private key. The second calculation is the valid time-frame for the new/current private key, as an extension to the previously calculated time-frame. Node B will realise that it did not receive the $3^{rd}$ message from A's first private key successfully, and hence, factor this in when calculating a time-frame for node A's subsequent private key. Using the calculations above, B calculates the following:

$$T^0_{cut\_off} = 4.2 + ((2.0 + 0.25) \cdot 1)) = 6.45$$

$$T^1_{key\_valid\_time} = 6.45 + ((2.0 + 0.25) \cdot 3) = 13.2$$

This procedure continues until a new key chain is generated by node A, and node A distributes its new public key, after which the time validity process repeats itself. Whilst the time precision - regarding the calculation of a time-frame for incoming messages - is not exact, this approach prevents an adversary from forging signatures using old, expired private key values that the adversary has obtained.

It should be noted that, for this time validity calculation, we assume only regular and MPR OLSR nodes in the network - as is the case in our testbed. If nodes that produce additional message types, such as MID nodes, are present, one should factor in these message interval times to calculate a more realistic time-frame for incoming messages from such nodes. In this work however, it is assumed that only MPR nodes will perform additional responsibilities via TC message distribution, and hence we simply use HELLO message intervals to calculate an upper-bound validity time for a HORS private key.

***Processing OLSR Packets***

The procedures involved with handling incoming and outgoing OLSR packets differ to that of the IBS signature scheme method due to the added complexities involved with the HORS key distribution requirements, and the use of two signature schemes. Figure 3.21 shows an overview of procedures and decisions involved with incoming OLSR packets, and Figure 3.22 for outgoing packets.

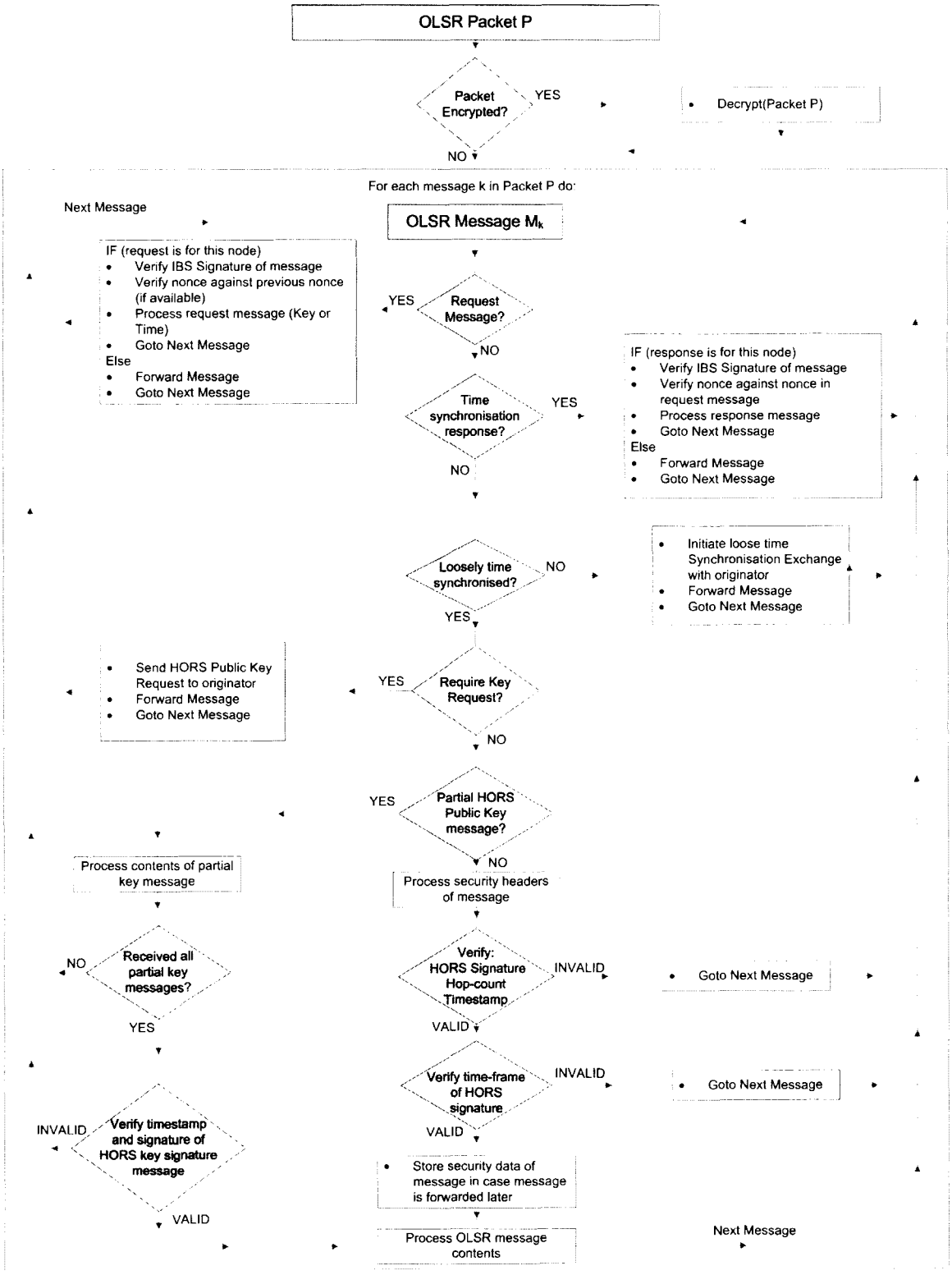



Figure 3.21: Processing incoming OLSR packets under the Hybrid signature approach

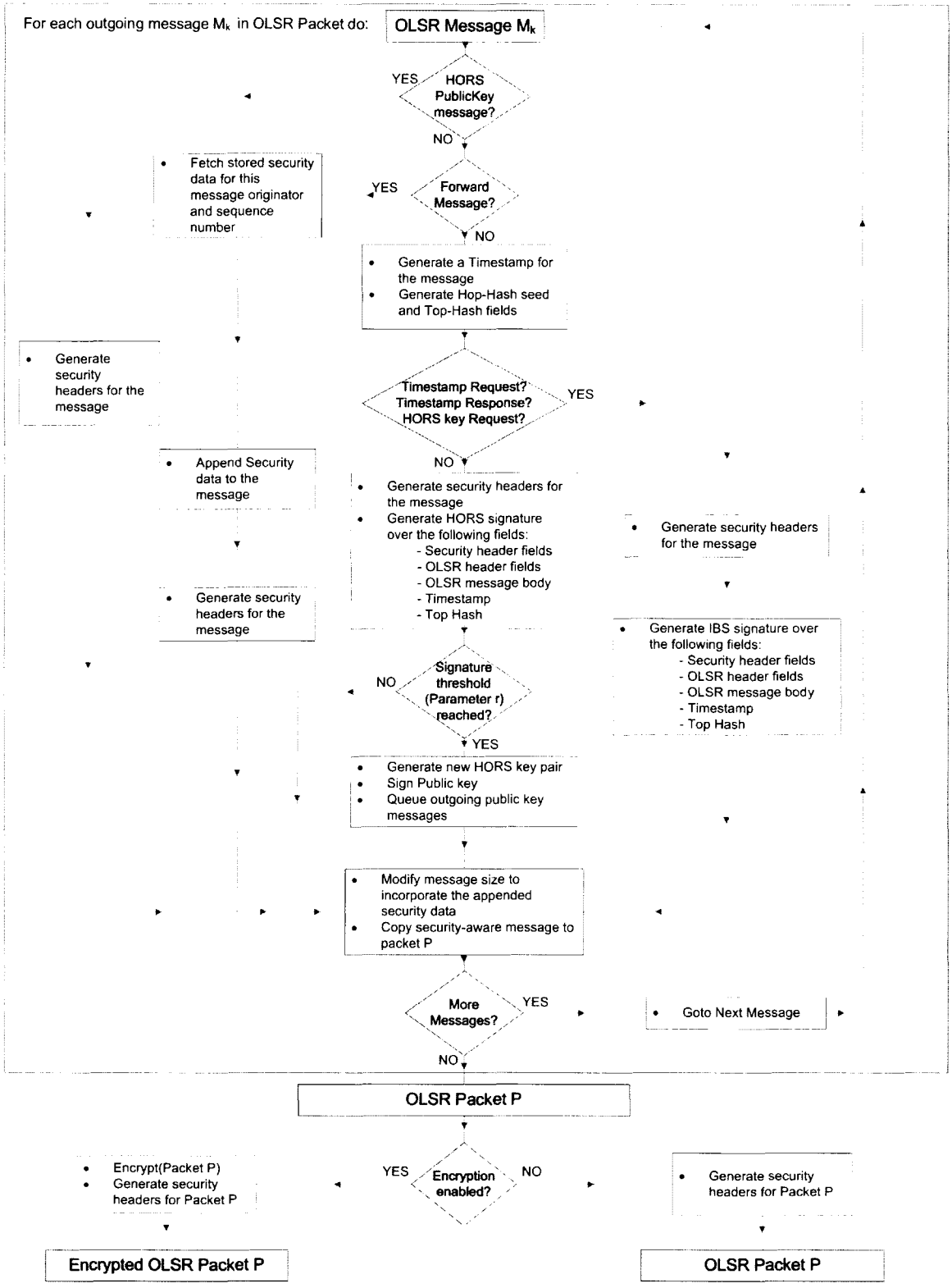



Figure 3.22: Processing outgoing OLSR packets under the Hybrid signature approach

CHAPTER 4

# IMPLEMENTATION

## 4.1 INTRODUCTION

To perform experiments on a live testbed in order to determine the impact the proposed security has on the OLSR protocol, an implementation of OLSR that incorporates the security mechanisms discussed in chapter 3 is required. To achieve this, the following components have been implemented as part of this work:

- Shamir's Identity Based Signature scheme based on RSA.
- The HORS one-time signature scheme, as well as the extensions to the scheme discussed in section 3.5.1.
- A secure OLSR implementation incorporating the security mechanisms discussed in chapter 3.

In this chapter, the details of implementing each of the components listed above will be elaborated on. Prior to discussing the implementation details of each of these components, we introduce the common implementation tools which have been used throughout the implementation phase.

## 4.2 IMPLEMENTATION TOOLS

### 4.2.1 The Linux Operating System

The Gentoo Linux™distribution [22] (kernel 2.6) was used throughout the implementation phase as a development Operating System (OS). The Linux operating system was chosen as it is open source and takes advantage of the plethora of existing open source applications and tools.

### 4.2.2 The GCC GNU Compiler Collection

The C programming language was used as it was most compatible with the other software that had been used throughout the implementation phase of this work: an OLSR implementation and

a big number library, both used as a basis for our security implementation, have been developed in C. GCC, the GNU Compiler Collection [19], was used as a C development platform.

### 4.2.3 The MIRACL Library

A big number library known as the Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL) [57] was used throughout the implementation of this work. The MIRACL library provides a set of implemented primitives which allow further cryptographic systems to be developed upon. The MIRACL library was used extensively in this work to provide the following features:

- Big Number Support: In some cases, the data types provided by a programming language are insufficient in size for a particular operation. MIRACL provides the ability to use big numbers, and perform operations with big numbers as if they are regular variables.
- Power and Modulo Arithmetic of large numbers.
- Hashing and Cryptographic Primitives: The MIRACL library provides implementations for various security primitives. In this work, the SHA-1 [47] hash and AES encryption/decryption routines provided by MIRACL have been used.

### 4.2.4 Valgrind

Valgrind [64] is a suite of tools that provide debugging and profiling facilities for programs developed in Linux. Memcheck, a memory management tool from the valgrind suite, provides detection of memory management problems for programs developed in C/C++. The Memcheck tool was used primarily to detect memory leaks when developing the security extension for OLSR; this ensures that no memory leaks occur during the execution of our secure OLSR implementation.

## 4.3 Implementation Method

To produce a working version of OLSR that incorporates the security design from chapter 3, the following two options are available:

1. A complete OLSR implementation: One approach would be to implement the entire OLSR routing protocol whilst also incorporating our security mechanisms. However, this approach does require a lot of redundant work that has already been done in existing OLSR implementations.
2. Adapt an existing implementation: The second approach would be to use an existing implementation of the OLSR protocol as a base for adaption. The implementation would have to be adapted in various ways to incorporate the security mechanisms discussed in the design chapter. This approach requires less work than the first approach, however, adaptations to existing implementations may not always be possible, and can in some cases produce an unstructured program.

### 4.3.1 OLSR Implementations

Currently, there are several available implementations of the OLSR routing protocol [62, 1, 6, 14]. Some of these implementations exist for simulation purposes, whereas some exist for practical use in a live ad hoc network. Owing to the fact that our experiments would be performed on a wireless testbed, the focus fell on implementations for the Linux operating system. The following are two popular and commonly used OLSR implementations:

- The OOLSR Implementation from INRIA [1]: OOLSR is a complete re-implementation of OLSR in the C++ programming language. OOLSR is fully compliant with the OLSR RFC (RFC 3626) [11]. The implementation can also be used with the NS-2 simulator.
- The OLSRd Implementation from UniK [62]: The OLSR daemon (OLSRd) is a multi-platform implementation of the OLSR routing protocol, developed in C. The OLSRd implementation is very modular and supports the use of plugins, making it very extensible.

The OLSRd[1] implementation was chosen as a basis for our security extension due to its modular design and support for extended functionality via dynamically loaded plugins. Instead of having to modify the original source of the software, the use of dynamic plugins is ideal for adding security functionality. The goal of our implementation was to develop a security module within OLSRd (shown in figure 4.1) which would provide the necessary security mechanisms, previously discussed in chapter 3. The following section first provides a brief overview of the OLSRd implementation, and then discusses the extensions that have been implemented as part of this work.

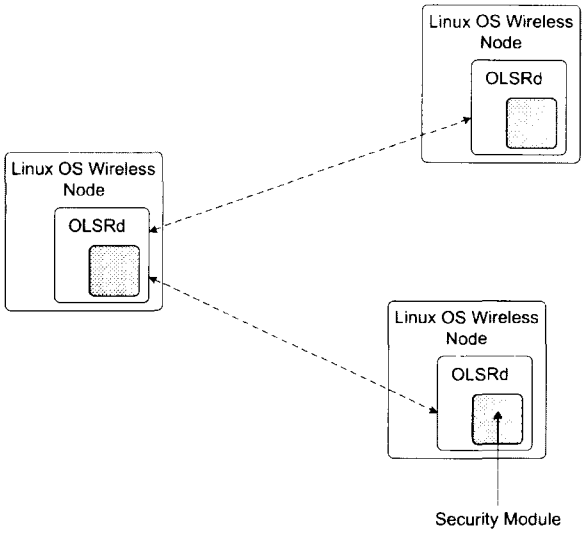



Figure 4.1: Security Module for the OLSRd Implementation

[1] Version olsrd-0.4.10 of OLSRd has been used throughout the implementation phase

## 4.4 THE OLSRD IMPLEMENTATION FROM UNIK

The OLSR daemon is a complete implementation of the OLSR routing protocol and is fully compliant with the OLSR RFC specifications. OLSRd is a very modular implementation and consists of several main components. A detailed explanation of the OLSRd implementation is given in [61]; in this section however, we provide an overview of the OLSR daemon, and highlight on its components that are relevant to this work. We also provide an overview of how dynamic plugins can be implemented.

### 4.4.1 The Socket Parser Component

OLSRd requires the use of network sockets to allow communication between OLSR nodes. The OLSRd implementation maintains a component, known as the socket parser, which is responsible for listening to traffic on all sockets that have been registered with the socket parser. At any time, a socket can be registered or deregistered dynamically with the socket parser. Once data has been read from a socket, the data needs to be processed in some specific way. To achieve this, OLSRd requires that a function be registered to a socket whenever a socket is registered with the socket parser. Therefore, if data is read on a registered socket, the data is passed on to the corresponding function of the socket. OLSRd exports the following function calls which allow sockets to be registered or deregistered.

OLSRd Socket Parser Functions

```
int add_olsr_socket(int, void (*)(int));

int remove_olsr_socket(int, void (*)(int));
```

The first parameter is a file descriptor for the socket that is going to be registered with the socket parser. The second parameter is a pointer to the function that will process the incoming data on the socket being registered.

### 4.4.2 The Packet Parser

By default, OLSRd receives OLSR data via the default OLSR socket (UDP port 698) that is registered with the socket parser. Once data has been read from the default OLSR socket, it is interpreted as an OLSR packet. A parser module is then responsible for determining how each individual message within the packet should be processed. This parser module is known as the packet parser. The packet parser allows message types to be registered with OLSRd; associated with each registered message type is a processing function, this function is responsible for processing any incoming messages it corresponds to. Message types, and their corresponding functions, can be registered/deregistered with the packet parser dynamically.

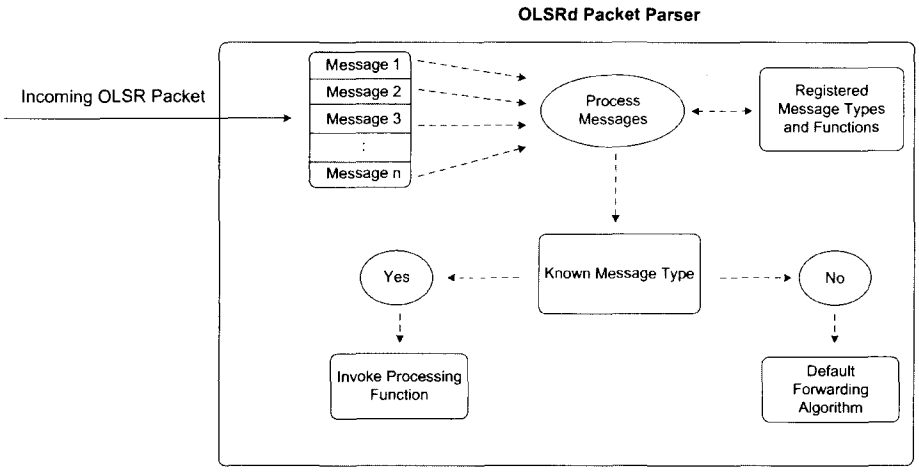



Figure 4.2: The OLSRd Packet Parser.

Figure 4.2 shows the general structure of the packet parser component of OLSRd. When an incoming OLSR packet is received, the packet parser processes each OLSR message and hands the message to its corresponding function. If no function is registered for a given message type, the message is simply forwarded according to the default OLSR forwarding algorithm. The two main functions governing the operations of the OLSRd packet parser are listed below.

OLSRd Packet Parser Functions

```
int olsr_parser_add_function(void (*)(union olsr_message *,
                                      struct interface *,
                                      union olsr_ip_addr *),
                             olsr_u32_t,
                             int);


int olsr_parser_remove_function(void (*)(union olsr_message *,
                                         struct interface *,
                                         union olsr_ip_addr *),
                                olsr_u32_t,
                                int);
```

The first parameter is a pointer to a function that will process the specified type of message. The function itself takes as parameters the message that needs processing, and the interface and IP address the message was received from (this is not necessarily the originator of the message). The 2nd parameter[2] is a value between 0-255 which represents the message's type, and the final parameter indicates whether the function performs forwarding of the message.

[2]olsr_u8_t, olsr_u16_t, olsr_u32_t are C type definitions used by the OLSRd implementation to represent unsigned integers of size 8, 16, and 32 bits respectively

### 4.4.3 Packet and Message Format in OLSRd

As previously mentioned, OLSR requires the use of many different default message types. Over and above this, OLSR allows for custom message types to be defined. Message types 0-127 are reserved for default message types (HELLO, MID, etc.) and for any future extensions [11]. The values 128-255 are reserved for private use. OLSRd uses a struct to define the OLSR message format (see appendix C.1). The first 7 variables of the struct represent the headers of the generic OLSR message format. The union is used to represent any one of the default OLSR message types (hello, tc, hna, mid). A union *message* represents the message body and will differ depending on the message type being carried.

### 4.4.4 Information Repositories

OLSR requires each node to maintain information regarding links, neighbours, and routes. OLSRd uses a double linked-list indexed by a static array for each information repository it requires; the array is accessed using a hash-table lookup mechanism. The reason behind such a choice of data structure is two-fold: 1) a double linked-list is a trivial and suitable data structure for storing data sets of a dynamic size; and 2) a static hash-table lookup mechanism allows for data to be located quickly, this is very useful as OLSR nodes require regular updating and accessing of routing information. To lookup information for a specific node, OLSRd uses a hash of the node's IP address as a lookup key for the static array. Currently the default hash operation used by OLSRd is simply the lowest 5 bits of an IP address; this allows for 32 unique hash values. The first element of every double linked-list is an empty root element, both for lookup purposes and to indicate the beginning and end of a double linked-list. No data is ever stored in the root elements. The information repository data structure of OLSRd is shown in figure 4.3.

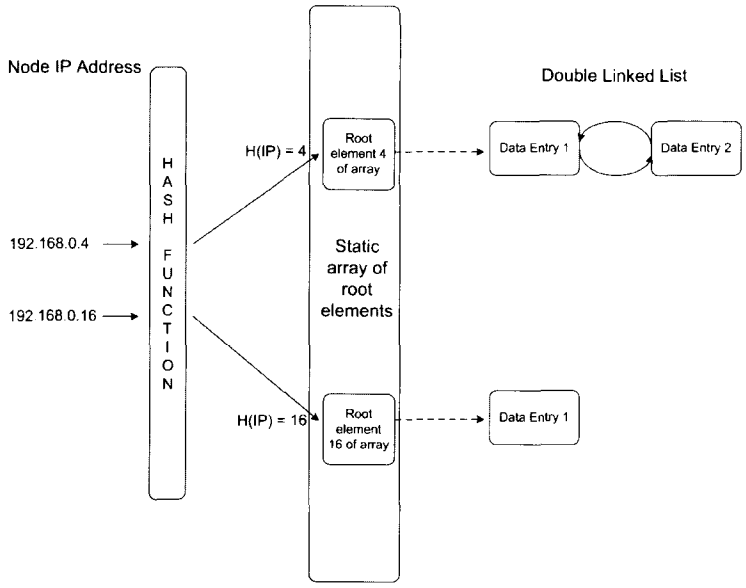



Figure 4.3: The OLSRd Information Repository Structure.

### 4.4.5 Scheduler

OLSR requires that information be kept fresh and up-to-date. OLSR also requires the execution of certain events at regular intervals; the generation of messages at their specified intervals is just one example of such events. To perform such periodic operations, OLSRd depends heavily on its scheduler component. The scheduler polls according to a specified interval, by default 0.1 seconds. The scheduler allows for two different types of functions to be registered with it: 1) functions that must be executed at a constant, regular interval; and 2) functions that must be executed at every scheduler poll, OLSRd calls this a timeout function. OLSRd provides the following two function calls to allow scheduled functions to be registered:

OLSRd Scheduler Functions

```
int olsr_register_scheduler_event(void (*)(), float, float, olsr_u8_t *);

int olsr_register_timeout_function(void (*)());
```

The first parameter of both function calls is a pointer to the function that must be executed by the scheduler. For the scheduler event registration function, the second parameter specifies the execution interval in seconds, the third parameter is an interval (seconds) after which the first execution of the function must occur, and the fourth parameter is a trigger which can force the scheduled function to be executed at the next poll.

### 4.4.6 OLSRd Configuration File

The OLSRd implementation makes use of a configuration file at startup time to register various parameters for the execution of the OLSR protocol. The following list represents some of the main features that can be altered from the configuration file:

- Interval Times: Message interval times can be specified in the configuration file.
- Network Interfaces: The configuration file can be used to specify which network interfaces OLSRd operates on.
- Dynamic plugins: The configuration file is used to specify which plugins must be loaded by OLSRd at execution time. Individual parameters for plugins can also be specified in the configuration file. The parameters will be loaded by OLSRd and passed to their respective shared objects (dynamic plugins).

## 4.5 SECURITY PRIMITIVES AND MECHANISMS

The secure OLSR protocol proposed in chapter 3 requires the use of various security primitives and mechanisms, namely:

**1. One-way Secure Hash Function**

As already mentioned, the MIRACL SHA-1 secure hash function implementation has been used throughout this work.

**2. Timestamp Mechanism**

The GNU C Timeval structure is used as a timestamp to represent an accurate point in time. The timeval structure contains two long integer values: the first representing the number of seconds since the epoch; the second value represents the number of microseconds. The following GNU C function is called to construct a timeval structure with the current calendar time:

```
Function int gettimeofday (struct timeval *tp, struct timezone *tzp);
```

Timestamp information is returned through the first parameter, a pointer to a timeval structure. Timezone information can be returned through the second parameter, however this can be left as NULL.

**3. Signature Schemes**

The IBS signature scheme and extended HORS signature scheme have been implemented as part of this work. The implementation details of the signatures schemes are explained in the remainder of this chapter.

### 4.5.1 The HORS Signature Scheme

The HORS one-time signature scheme, including the extensions discussed in section 3.5.1, has been implemented in C as a stand-alone, separate component. Apart from implementing the algorithms involved with this signature scheme, the main implementation decisions surrounded the data structures involved with managing the keys.

***HORS Key Data Structure***

Owing to the fact that various parameters for the HORS signature scheme are initialised from a configuration file at run-time, the size of the HORS keys, and hence the data structures being used, cannot be set at compile-time. This effectively means that memory is allocated dynamically at run-time, according to the parameters specified.

For a HORS key - public or private - a MIRACL BIG number is used to store each value of the key. Assuming the use of the SHA-1 hash function, the largest possible size of a single value in a key would be 160 bits. To store a single HORS key, whether it be a private or public key, a block of memory on the heap is allocated for $t$ contiguous BIG numbers, representing all the

values from the key. The elements of a key can be accessed like an array, using the memory pointer as a starting position to the array.

### *HORS Key Chain Data Structure*

To store multiple HORS keys in a key chain structure, a 2-dimensional pointer storage structure is used. A contiguous block of memory is allocated for a specified number of lookup pointers, the number being dependent on the size of the key chain. Each lookup pointer is just a pointer to the first value of the key it corresponds to. For instance: the very first lookup pointer will point to the very first HORS key in the key chain. Using this approach allows each key in the key chain to be indexed quickly, and it still allows the size of the structures to be specified dynamically at run-time via the configuration file. The structure used to store the key chain is shown in figure 4.4.

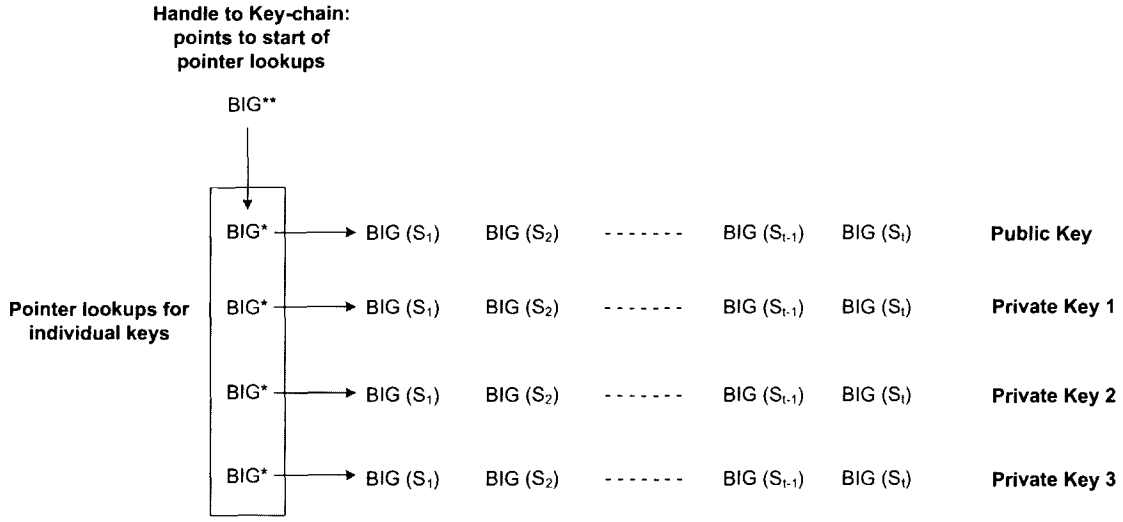



Figure 4.4: The structure of a HORS key chain using the C language.

### *HORS Component Interface*

The HORS signature scheme component exports the following main function calls:

HORS Component Interface

```
int initialise(int k, int l, int t, int r, int key_chain_size);

int generate_key_chain( );

int construct_signature(char *data, int data_size, char *signature);

int verify_signature(char *data, int data_size, char *signature,
                     BIG *public_key,
                     int hash_iterations);
```

The initialise call initialises the HORS component with the parameters $k, r, t, l$ and requires the size of a HORS key chain be specified. The size of a key chain is specified in terms of the number of total keys - public and private - in a key chain. Parameters to the verify function call are: a pointer to the data the signature is created over; the size of the data; a pointer to a block of memory which serves as the signature to be verified; the HORS public key the signature must be verified against; and an integer representing the number of times each signature value must be hashed, in order to compare the values against the public key.

## 4.5.2 The Identity Based Signature Scheme

Like the HORS software component, the Shamir IBS signature scheme has been implemented using the MIRACL library. The MIRACL library provides the ability to perform power and modulo operations on large numbers, this is required when implementing the signature and verification operations of the IBS scheme, as explained in section 2.7.1.

### *IBS Key Generation*

The MIRACL library comes bundled with sample programs for various security and signature operations. One of these programs is a simple implementation of RSA and provides RSA key generation. The MIRACL RSA implementation was used to generate a RSA 1024-bit public/private key pair. The RSA public and private keys then serve as master public and private keys for the IBS key generation and signature verification operations. To generate the IBS private keys, a program has been developed which takes a command line parameter representing the IP address of a node; the IP address is given as four bytes in decimal notation, as follows:

```
./generate_ibs_private <IP address>

./generate_ibs_private 192 168 0 10
```

The second line above shows the private key program being executed for the IP address *192.168.0.10*. The program uses a SHA-1 hash of the 32-bit IP address as the IBS public key for the node; the IBS public key is then used with the master private key (RSA private key) to generate an IBS private key which corresponds to the IP address. The IBS private key is then written as output to a file. This program was used to generate an IBS private key for each OLSR thin-client node involved in the experiments.

***IBS Component Interface***

The IBS signature scheme component exports the following main function calls:

IBS Component Interface

```
BIG generate_public_key(unsigned int address);

int construct_signature(char *data, int data_size, char *signature);

int verify_signature(char *data, int data_size, char *signature);
```

The first function call is used to generate a public key for a given node. The function requires a 32-bit integer representing the IP address of a node. The function then performs a SHA-1 hash of the integer parameter and returns the 160-bit output as a MIRACL big number, representing the identity-based public key of the node in question. The signature construction and verification functions take as input: a pointer to the data the signature is created over; the size of the data; and a pointer to a block of memory which serves as the signature to be created or verified.

## 4.6 THE END-TO-END SECURITY PLUGIN FOR OLSRD

To provide OLSR with the end-to-end security mechanisms from our design (chapter 3), an OLSRd security plugin has been implemented as part of this work. In the design chapter, two different methods of providing security for the OLSR protocol, involving two different signature schemes, are discussed: the IBS signature scheme approach (see section 3.7.1); and a hybrid signature scheme approach (see section 3.7.2). The security plugin implemented as part of this work provides both methods of securing OLSR. This remainder of the section provides details of the security plugin implementation.

### 4.6.1 Plugin Initialisation

The plugin performs three main operations at initialisation time: parameter initialisation; key initialisation; and registration of message types and functions.

### *Parameter Initialisation*

The security plugin requires several parameters that are registered at run-time via the OLSRd configuration file. The following parameters are used to govern the operation of the security plugin:

- Encryption: value specifying whether encryption is to be used on OLSR packets.
- Signature scheme: IBS signature scheme or Hybrid signature scheme.
- HORS parameters $(k, t, l, r)$: various parameters used for the HORS signature scheme.
- HORS key chain size: total number of keys in an extended HORS key chain.

### *Initialisation of Security Keys*

An OLSR node using the IBS signature scheme requires an IBS private key that corresponds to the IP address of the node. To facilitate this requirement, a thin-client node accesses its pre-shared IBS private key from a mounted Network File System (NFS) directory on the central server. Although this is not practical in a realistic environment, the practice of secure key management in an ad hoc network is not the focus of this work.

### *Message and Function Registration*

The implementation requires several custom OLSR message types. The new message types are to be registered with OLSRd at the initialisation time of the plugin. Message type registration in OLSRd is handled using calls to the packet parser, explained in section 4.4.2. The following list contains the custom message types used in the implementation (see appendix C.2 for detailed formats of the custom message types):

- Time Synchronisation Request - OLSR message type 231
- Time Synchronisation Response - OLSR message type 232
- HORS Public Key Request - OLSR message type 233
- HORS Public Key Message - OLSR message type 234

The end-to-end security plugin also requires the registration of several functions with OLSRd. Each function and its purpose is explained below:

- HORS Key Chain Creation Function: A scheduled function to generate a new HORS key chain and distribute the new public key. This function is only executed via means of a trigger.

- Cleanup Function: function to check for expired data in the repositories and perform a cleanup. Timestamps associated with the data are used to determine how old the data is.
- Topology Change Function: function registered with OLSRd to execute when there is a change in the topology. This function is used to perform loose-time synchronisation when the hop-metric of a route to a node changes.

### 4.6.2 Information Repositories

The structure of the information repositories used in the security plugin is the same as that used by OLSRd. Three additional information repositories are required to be maintained by each OLSR node for security purposes.

***Security Details of Neighbours***

This table maintains security details specific to other OLSR nodes in the network. The table below gives an overview of the details maintained in this repository and the data type/structure used to represent each detail in the repository.

| Detail | Data Type/Structure | Summary |
|---|---|---|
| IBS public key | MIRACL BIG | 160-bit SHA-1 hash of a node's IP address. Used for verification of IBS signatures |
| $T_R$ | GNU C timeval | Timestamp generated at the time of sending a loose time synchronisation request. Required for timestamp verification of incoming OLSR messages |
| $RTT$ | GNU C timeval | The Round-trip time calculated during the loose time synchronisation process. Required for timestamp verification of incoming OLSR messages |
| Expiry timestamp | GNU C timeval | Expiry timestamp indicating when the data corresponding to this neighbour should be removed |
| Nonces | 32-bit Integers | Nonces are used for the loose time synchronisation and HORS public key request messages. Nonces provide replay attack protection for the request messages, as well as time synchronisation response messages |

Table 4.1: Security Details of Neighbour Nodes

### *HORS Public Key Table*

This table is where an OLSR node stores HORS public keys that have been successfully received from neighbouring nodes. A HORS public key is only successfully received once all the packets carrying the key have been received successfully, and the signature and timestamp created over the packets verifies correctly.

| Detail | Data Type/Structure | Summary |
|---|---|---|
| HORS public key | MIRACL BIG * | HORS public key of a neighbour node |
| HORS key sequence | 16-bit Integer | Last recorded key sequence of a received HORS public key |
| Index of last private key verified | Integer | Represents the index of the last private key - corresponding to this public key - verified by this node. Used in calculating the remaining validity time of a HORS private key |
| Signatures left | Integer | Indicates how many signatures can still be provided by the current private key of the neighbour in question. Used in calculating the remaining validity time of a HORS private key. |

Table 4.2: HORS Public Key Table

### *Message Security Table*

As a message traverses the network, the security data originally constructed for the message is required at each hop. This requirement enables a security association to exist between a message originator and a message destination. If a message is received by a node and the message is due to be forwarded, the security data associated with the message is stored temporarily and then transmitted with the message when it is forwarded. An OLSR node is required to maintain a table for the temporary storage of security data associated with a message that is due to be forwarded.

| Detail | Data Type/Structure | Summary |
|---|---|---|
| Message sequence number | 16-bit Integer | Sequence number of the message the security data corresponds to |
| Message timestamp | GNU C timeval | Timestamp of the message |
| Top-hash | 20 byte/char array | The top-hash of the message |
| Signature | byte/char array | The signature constructed over the message. The signature data can represent a HORS or IBS signature |
| Hop-hash | 20 byte/char array | Hop-hash data corresponding to the current hop-count of the message |

Table 4.3: Message Security Table

### 4.6.3 Incoming OLSR Packets

Figures 3.13 and 3.21 in chapter 3 provide a detailed overview of the process involved with processing a secure incoming OLSR packet. This section provides an overview of how the security plugin interacts with OLSRd in order to perform the required security operations on incoming OLSR packets. OLSRd by default, through means of the socket parser, processes OLSR packets through a registered socket and a corresponding processing function. In order to process an altered, unconventional OLSR packet - as is the case with a secure OLSR packet - the OLSR packet must be intercepted prior to being passed to OLSRd for conventional processing. To achieve this, the security plugin deregisters the default OLSR socket with the socket parser, and registers a socket with a new processing function.

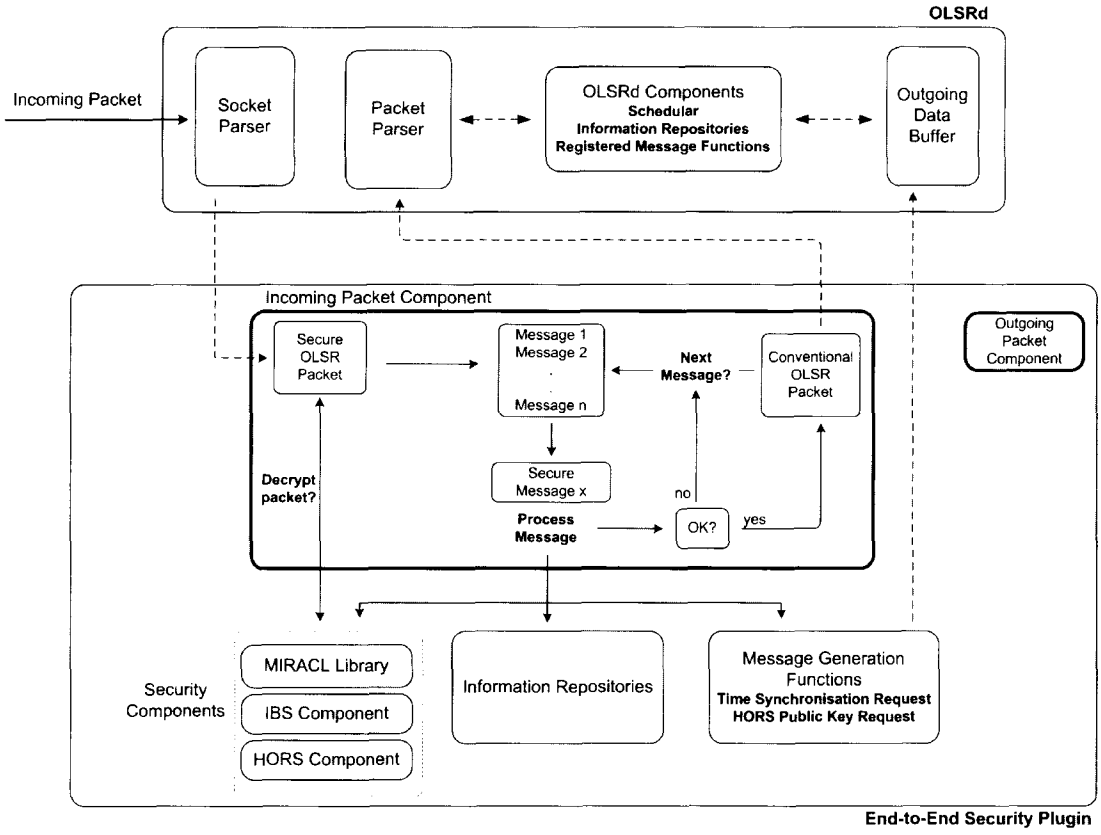



Figure 4.5: Processing incoming OLSR packets - end-to-end OLSRd security plugin

Figure 4.5 illustrates how the plugin intercepts incoming OLSR data prior to the data being passed to the packet parser for processing. Once OLSR data has been intercepted, the security plugin processes the OLSR packet and the individual messages carried within the packet, as per our design. If a message verifies correctly, the conventional OLSR headers and the message body (without any security data) is copied to a second buffer; if the message is due to be forwarded, the security data corresponding to the message is stored temporarily in the information repository. Once all messages have been processed, the second data buffer acts as a conventional OLSR packet. The packet headers are constructed to reflect the size of the new packet, and the packet is handed back to the packet parser component for conventional processing.

Whilst processing incoming OLSR control data, loose-time synchronisation and HORS public key requests may have to be queued for transmission.

### 4.6.4 Outgoing OLSR Packets

To secure an outgoing OLSR packet and each message carried within the packet, it is required that the security plugin intercepts the OLSR packet prior to it being transmitted over a network interface. To achieve this, the plugin registers a Packet Transform Function (PTF) with OLSRd. OLSRd makes use of an outgoing data buffer which buffers outgoing OLSR messages prior to transmitting the messages. Just prior to an OLSR packet being transmitted, a handle, or pointer in programming terms, of the outgoing data buffer is passed to registered packet transform functions. The packet transform function, registered by the security plugin, operates on an outgoing OLSR packet as per our design, shown in figures 3.22 and 3.14. The interactions between the plugin and OLSRd is shown in figure 4.6.

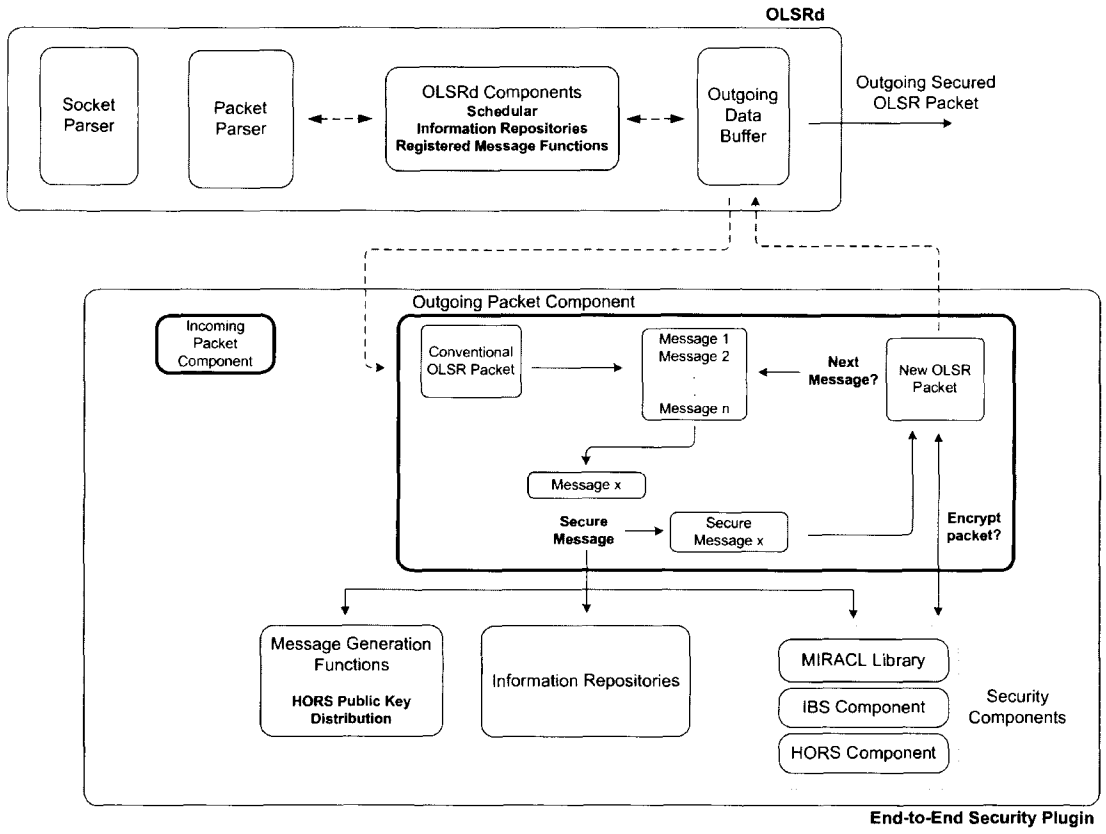



Figure 4.6: Processing outgoing OLSR packets - end-to-end OLSRd security plugin

Once the packet transform function has completed processing the outgoing OLSR packet, the modified data buffer is passed back to OLSRd for transmission. Take note, while outgoing messages of a packet are being signed using extended HORS, it is possible for the HORS key chain to expire. Hence, a new key chain may need to be generated and its public key distributed - this is indicated by the interaction between the outgoing packet component and the message generation component in figure 4.6.

## 4.7 CONCLUSION

In this chapter, we first identified what components required implementing, and then discussed the implementation details of each component. Rather than implementing an entire version of OLSR with our security mechanisms, we decided to use the OLSRd implementation as a basis for adding security functionality. The security functionality was added as a security plugin for OLSRd, through means of a dynamically linked library - or shared object in Linux terms. The signature schemes used in this work were implemented as separate components, and utilised by our OLSRd security plugin. In this chapter we also discussed the common implementation tools that were necessary throughout the implementation phase.

CHAPTER 5

# WIRELESS MESH TESTBED

A hardware testbed was used for the experiments undertaken as part of this work. The wireless mesh tested consists of a total of 9 wireless terminals. This chapter focuses on the details of the wireless testbed, and gives an overview of the specifications of the wireless terminals forming the testbed.

## 5.1 WIRELESS NODES

Each wireless node in the testbed is a Mini-ITX system. Table 5.1 below, gives an overview of the specifications of the Mini-ITX systems used in the testbed.

| Processor Type | Processor Speed (MHz) | Physical Memory (RAM) | Onboard Networking |
|---|---|---|---|
| Via C3 Nemiah | 1000 | 128 MB | Ethernet 100 Mbps |

Table 5.1: Mini-ITX Technical Specifications

The Mini-ITX systems do not have hard disks for persistent data storage. Instead, the systems run from a RAM disk.

## 5.2 WIRELESS NETWORK INTERFACE CARDS

A wireless Network Interface Card (NIC) had to be installed, as an additional peripheral, in each node. The SMC SMCPCIWT-G wireless NIC, based on the Atheros chipset, is used as a wireless network interface for each node in the testbed. The SMCPCIWT-G wireless NIC is IEEE 802.11b/g compatible, and supports speeds of up to 54Mbps. Open-source drivers provided by MadWifi - a team developing Linux kernel drivers for WLAN devices with Atheros chipsets - have been used for the wireless NICS.

## 5.3 ANTENNA AND ATTENUATOR

When considering an indoor wireless mesh testbed, that consists of several nodes communicating in a small area, a challenge that exists is limiting the signal between the nodes to create a multi-hop environment. For instance, the wireless signal of a node should reach a neighbour node A, classified as 1 hop away. However, the signal should not reach the neighbour of node

A, classified as 2 hops away, thus, creating a multi-hop path via node A. To limit the signal strength, created by each wireless card, attenuators have been used. The attenuators are present between the wireless NIC and the antenna of each wireless node.

The antenna of each wireless node is raised approximately 80cm off the ground. This was done to prevent the metal chassis of each node from interfering with the propagation of the wireless signals. Figure 5.1 shows a complete Mini-ITX system with a raised antenna.

Figure 5.1: Wireless client node with a raised antenna.

## 5.4 Operating System

The operating system for each Mini-ITX system consists of:

- Gentoo Linux 2.6 Kernel: A manually compiled/configured kernel. Kernel support for hardware/software has been kept to a minimum, in an attempt to keep the kernel as compact as possible.
- Embedded Root Filesystem: A lightweight base-layout gentoo filesystem, with minimal packages installed. The core packages required on each node include[1]:
  - busybox: lightweight version of coreutils
  - dropbear: lightweight SSH daemon. Allows remote login to the wireless nodes via SSH.
  - net-tools: network administration tools

[1] All packages for the client nodes have been compiled against uclibc (a tiny version of GNU C Library))

- wireless-tools: wireless networking support
- udhcpcd: small DHCP client

As already mentioned, the Mini-ITX systems do not maintain hard disks. Instead, each systems runs from a RAM disk. Hence, keeping the size of the kernel and filesystem to a minimum is a priority. The smaller the size of the kernel and embedded filesystem, the greater the amount of physical memory available for processes running on the nodes.

The kernel and embedded root filesystem is loaded into RAM at boot time. To achieve this, the Preboot Execution Environment (PXE) protocol is used. PXE protocol support is offered by most NICs, and allows a system to boot directly off a network from a PXE server (commonly known as netbooting). At boot time, a client system performs network configuration using DHCP with a central controller. During this process, PXE configuration options are also exchanged from the controller to the client. Once complete, the client can attempt to boot by downloading the kernel and filesystem from the central controller.

## 5.5 TESTBED ARCHITECTURE

Figure 5.2 shows the architecture of the mesh testbed. The central controller acts as a DHCP server, PXE boot server, and also a MySQL database server for testing purposes. Each wireless node is connected with the central controller via ethernet. When performing experiments, the nodes communicate with each other via Wi-Fi in ad hoc mode.

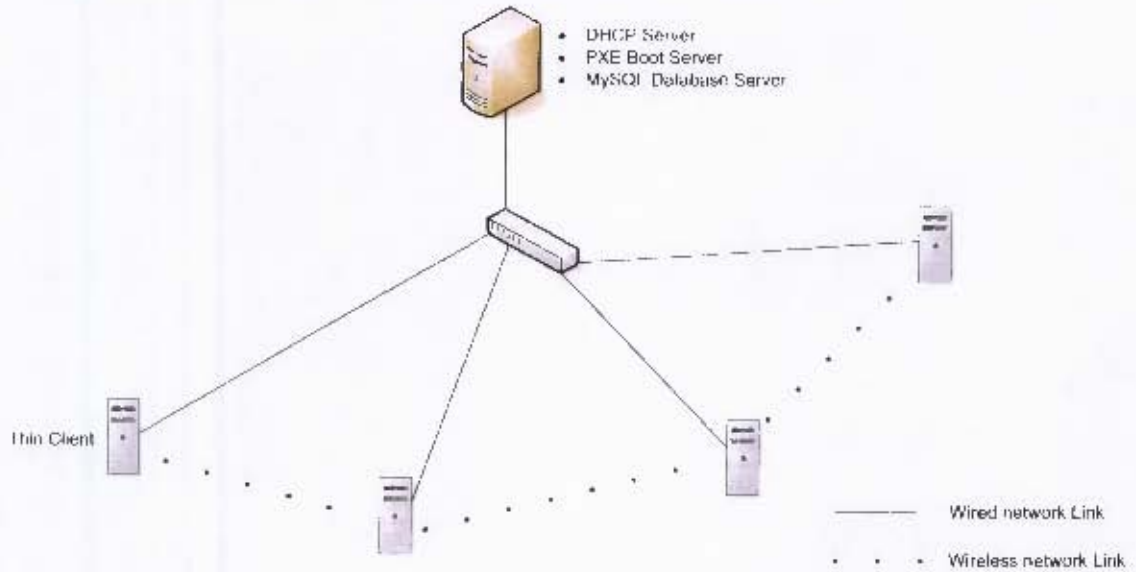



Figure 5.2: Wireless Testbed Architecture.

## 5.6 CONCLUSION

In this chapter, the wireless ad hoc testbed used for the experimental part of this study has been introduced. The operation and architecture of the testbed, as well as the individual specifications of each node forming the testbed, has been discussed. The following chapter explains the various tests that were undertaken on the testbed, as the experimental part of this study.

CHAPTER 6

# TESTING - EXPERIMENTATIONS

## 6.1 INTRODUCTION

This chapter discusses the various tests that were performed as part of the experiments undertaken in this work. We first discuss the signature scheme tests that were performed to determine the signature creation and verification times of the signature schemes used as part of our OLSR security solution.

## 6.2 SIGNATURE SCHEME TESTS

Signature creation and verification times were obtained for the IBS and extended HORS signature schemes. The signature operations were performed on a thin-client node from our testbed; the hardware specifications of each thin-client is disclosed in the testbed chapter. The times obtained in our tests represent the elapsed time spent in the CPU whilst performing a signature operation. To obtain the CPU time for the signature operations, the GNU C *clock* function was used; the GNU C *clock* function can be used as follows, as explained in the GCC library documentation [20]:

GNU C CPU Time

```
#include <time.h>

clock_t start, end;

double cpu_time_elapsed;

start = clock();

        /* Perform the operation that must be timed */

end = clock();

cpu_time_elapsed = ((double) (end - start)) / CLOCKS_PER_SEC;
```

The time resolution of the clock function is not high enough to time a single execution of our signature operations. Hence, for these tests, timing was performed over 1000 iterations of each signature operation. The following algorithms and parameters were used in the signature scheme tests:

- The SHA-1 hash function (160-bit output) was used for all signature operations.
- The IBS signature scheme implemented as part of this work.
- Extended HORS signature scheme with parameters: $\{t, k, l\} = \{256, 20, 80\}$. We call this parameter set HORS 256.
- Extended HORS signature scheme with parameters: $\{t, k, l\} = \{1024, 16, 80\}$. We call this parameter set HORS 1024.
- HORS key chain with 91 keys - 1 public key; 90 private keys.

## 6.3 TESTING TOOLS USED

Apart from implementing our own testing mechanisms (explained in section 6.7 below), the following software tools have been used for the purposes of testing:

- Tcpdump: tcpdump is a network debugging tool. Tcpdump can be used to capture TCP/IP and UDP traffic being transmitted across a network.
- MySQL: The MySQL database is used to log data that is captured during the experiments.

## 6.4 TESTING TOPOLOGIES

To test how our security affects the performance of the OLSR routing protocol, several testing topologies will be used as a basis for comparison. The following testing topologies will be used:

1. An OLSR node with a direct neighbour (Dual).
2. An OLSR node with 2 direct neighbour (Triple).
3. A 3x3 arrangement of OLSR nodes (Grid).
4. A straight line topology of 9 OLSR nodes (Linear).

## 6.5 OLSR VARIATIONS TESTED

For each of these topologies, the following variations of OLSRd and our security plugin have been used for testing purposes:

1. The original OLSR protocol, no security extension.
2. Security-aware OLSR: IBS Signature scheme.
3. Security-aware OLSR: Hybrid Signature scheme (1)
   - Standard HORS 256: Parameters $\{t, k, l, r\} = \{256, 20, 80, 2\}$
     A single Public and Private key in a key-pair
   - Standard HORS 1024 - Parameters $\{t, k, l, r\} = \{1024, 16, 80, 6\}$
     A single Public and Private key in a key-pair
4. Security-aware OLSR: Hybrid Signature scheme (2)
   - Extended HORS 256 - Parameters $\{t, k, l, r\} = \{256, 20, 80, 2\}$
     Private keys in a HORS key chain - 8
   - Extended HORS 1024 - Parameters $\{t, k, l, r\} = \{1024, 16, 80, 6\}$
     Private keys in a HORS key chain - 11
5. Security-aware OLSR: Hybrid Signature scheme (3)
   - Extended HORS 256 - Parameters $\{t, k, l, r\} = \{256, 20, 80, 2\}$
     Private keys in a HORS key chain - 60
   - Extended HORS 1024 - Parameters $\{t, k, l, r\} = \{1024, 16, 80, 6\}$
     Private keys in a HORS key chain - 75

The value of parameter r for both HORS parameter sets (HORS 256 and HORS 1024) has been chosen such that both sets offer similar strength in security. Owing to the fact that the OLSR protocol is a broadcast protocol, and thus its data is one-to-many, signature verification time is more important than signature creation time. This fact makes selecting the number of private keys in HORS key chain quite important, as this has a direct impact on the signature verification time of the extended HORS signature scheme (see the signature scheme results in chapter 7). Variation 3 uses only a single public and private key per key pair, thus, making it the standard HORS signature scheme. For variation 4, the number of private keys in a key chain, for each parameter set, has been chosen using the RSA verification time (from the RSA implementation tested in chapter 7) as an upper-bound. For HORS 256, its verification time is similar to that of the RSA implementation tested, when the $8^{th}$ private key, from the key chain, has been used to create the signature being verified. For HORS 1024, it is the same case when the $11^{th}$ private key is used. For variation 5, the verification time of the IBS signature scheme was used as an upper-bound when selecting the number of private keys, in a key hash chain, for each parameter set.

For the hybrid-signature scheme variations listed above, a number has been associated with each in brackets. This number is simply to provide a means to distinguish between the 3 different variations, and is used when referring to the different hybrid-signature scheme variations.

## 6.6 PARAMETERS

### 6.6.1 Wireless Network Interface Properties

- Wireless Mode: 802.11b
- Data Rate: 1 Mbps (Megabits per second)

### 6.6.2 OLSR Parameters

For the experiments, the default OLSR parameters were used. The default message interval times and hysteresis values are listed below:

***Interval Times***

- HELLO message - 2 seconds
- TC Message - 5 seconds

***Hysteresis Values***

- High Threshold - 0.8
- Low Threshold - 0.3
- Scaling Factor - 0.5

### 6.6.3 Misc Parameters

The following parameter settings were used for the tests:

- Loose-time resynchronisation threshold:
  - Window period - 10 incoming messages
  - Invalid verification threshold - 3 failed verifications
- Loose-time verification - Slack: 0.5 seconds
- Valid time window for extended HORS incoming messages - Slack: 0.25 seconds per message
- HORS public key distribution window threshold - 2 seconds (HELLO message interval)
- Encryption - encryption was disabled for all the tests performed

## 6.7 OLSR TESTS - IMPLEMENTATION AND EXECUTION

The network topologies, as well as the different variations of OLSRd being tested have been listed in the section prior to this. However, tests need to be constructed in order to capture useful information from the software that is being tested. To capture data for some of the tests being performed: 1) mechanisms had to implemented into our security plugin to allow for such capture; and 2) a separate OLSRd plugin had to be developed to allow for such data capture when the standard OLSR protocol is being tested and no security is being used. Three different tests have been constructed and the purpose of each explained in the remainder of this section.

### 6.7.1 Ping Test

The ping test is used to establish a general round-trip time of an OLSR message (message ping time) across a varying number of hops. To perform this test, an OLSR ping custom message was created. The ping message contains a timestamp of the sender at the time of transmission; receiving nodes then resend the ping message as a reply. When the originator receives the reply, it can calculate a round-trip time for the message. Ping intervals can be specified in the OLSR config file under the ping plugin section. Two custom messages are used to perform the ping test:

- Ping Request Messages: A ping request message contains a timestamp as the message body. The timestamp is generated by the ping originator when the ping request message is constructed.

- Ping Response Message: A ping response message contains the original timestamp of the request message along with the IP address of the node responding. When the correct node receives the response, it can calculate the round-trip time using its current time and the timestamp of the response message. After a node calculates a ping time, it must then log the data to the MySQL database on the central sever.

Time-To-Live fields of the ping messages are set according to the role of the node sending these messages: an MPR node will set the TTL to its maximum value (255); a non-mpr node will set the TTL to 1. This is to prevent the ping test introducing additional time synchronisation and key requests which otherwise would not have been necessary.

### 6.7.2 Traffic Overhead Test

To perform the traffic overhead tests, the tcpdump utility program was used to capture traffic on the OLSR port. When performing a traffic analysis test, the tcpdump utility is started - listening for incoming and outgoing traffic on OLSR port 698 - prior to executing an OLSR test. Once complete, tcpdump can be used to obtain information regarding the traffic that was captured. Traffic analysis was performed on the most central nodes of the topologies described above in section 6.4.

### 6.7.3 Data Logging Test

The data logging test was used to capture information regarding various properties of the OLSR protocol and our security implementation. Each node logs data using a MySQL connection with the central sever.

The values captured from this test are grouped and classified according to different hop-counts. This allows one to determine average values for different hop-counts in the network. However, the unpredictable nature of wireless links can make the collection of accurate data, for different hop-counts, quite difficult. Hop metrics between nodes can change on a regular basis, therefore, a robust information gathering/logging mechanism is required. To achieve this, each node maintains a testing record for every other node in the network. A testing record, for a given neighbouring node, stores the information gathered from the tests, as well as the current hop-metric for that node. Testing data, for a given node, is dumped into the database at a regular interval, or, if the hop-metric to the node changes; this is required because the hop-metric of a node is closely related to the testing information that has been gathered. If the hop-metric of a route to a particular node changes, the testing record for the node in question must be updated to reflect the new hop metric accordingly; subsequent data collected will then be classified under the new hop metric.

Using a mechanism like this however can lead to inaccurate data if samples are not chosen correctly. For instance: node A has a testing record for another node B; A has a hop-metric of 2 hops for its route to node B. For every message A receives from B, node A will record information regarding the received message - as per the tests listed below. However, due to the unpredictable nature of wireless links, it is possible that a subset of messages received by node A, from node B, in fact traversed a number of hops different to that of 2. If this is the case, a recipient node (node A in this case) must neglect these messages from its test samples; this prevents these samples being included under the wrong hop-count when calculating averages under varying hop-counts. Therefore, a node must perform a check on an incoming message before it includes the message in its testing samples: the check ensures that the hop-count of the received message is equal to the current hop metric it has stored for the originator of the message. If the check is correct, the message is included as a test sample.

The information logged by each OLSR testing node is explained below:

***Delivery of OLSR Messages***

This represents the percentage of Successfully delivered OLSR control messages across a varying number of hops. To obtain such information, each node is required to log the OLSR sequence numbers of all the control messages it sends and receives, as well as the hop-count for each received message.

Once these sequence numbers have been logged, one can determine the percentage of successfully delivered OLSR messages by comparing sequence numbers between a given sender and receiver. Certain message types, however, are destined to traverse the entire network whereas other types, such as HELLO messages, only traverse a single hop. Therefore, it is not feasible to simply compare sequence numbers of two given nodes as these nodes may be several hops

apart, and hence, not receive each other's HELLO messages. To deal with this when recording sequence numbers, we group messages into two different types: HELLO messages, which travel only a single hop; and flood messages, intended to traverse the entire network via MPR distribution. We ignore sequence numbers of time synchronisation and key request messages, as these messages stop being forwarded once they reach the intended destination node.

***Route Delays***

Here we want to test the delay associated with establishing routes to all destinations, as well as the delay associated with repairing a route once it becomes broken or unavailable.

- Route Establishment: At initialisation time, a timestamp is calculated by each node. When an entry is inserted into a routing table for another node for the first time, a timestamp is calculated at this point; the difference between the latter timestamp and the initial timestamp is used as the delay to establish a route to the neighbouring node, at the specified hop-count.
- Route Repair: If an established route becomes unavailable, a timestamp is calculated at the point of the route being removed from the routing table. If the route is repaired, a second timestamp is calculated and the difference between the two timestamps is used as the route repair time.

Whilst this method may not be exact, it allows us to determine a rough average for these routing delays classified under different hop-counts.

***Route Availability***

This test aims to determine how available a route is, to a given node, once an initial route to the node has been established. To achieve this we use a polling mechanism; once an initial route to a node has been established, at every subsequent poll a check is made to determine if a route to the node still exists. At every poll a sample counter is incremented, and if a route does exist, an availability counter for the node in question is also incremented. The availability counter allows one to determine how available a route to a particular node is, relative to the total number of samples taken for that node.

***Dropped Messages***

Here we want to test the percentage of OLSR messages that are dropped due to our security requirements for the following reasons:

- Loose Timestamp Verification

  There are several factors that can affect the loose timestamp verification process: the round-trip time calculated can be inaccurate; and, unexpected transmission and processing delays of a message at each hop. This test allows us to determine if the loose timestamp verification is practical or if it results in too many messages being discarded. Here we also test the accuracy of the loose-time verification process.

- Extended HORS Time Validity Verification

  In section 3.7.2, a private key validity time for the extended HORS signature scheme is explained. Like the loose timestamp verification process, the HORS private key validity-time verification is also subject to unexpected transmission and processing delays of a message at each hop. By recording the number of dropped message we can determine the percentage of incoming messages dropped due to this check. The accuracy of the HORS private key validity time is also determined; this is achieved by recording the difference between the calculated key validity time frame for incoming messages, and the actual arrival time of the incoming messages.

### 6.7.4 **Performing the Tests**

For each testing topology, the 3 tests (ping, traffic analysis, data logging) were performed on each OLSR variation listed in section 6.4. For each variation tested, the ping and data logging tests were executed for a duration of 15 minutes, whereas the traffic analysis test was performed for 10 minutes; all 3 tests were executed twice to allow two separate sets of results to be gathered. The duration of 15 minutes is enough to allow the routes to converge and become stable, and allows enough data from each node to be collected. A script was used on the central server to control the execution of the tests remotely on the OLSR thin clients; each test was executed remotely on all the thin-clients at the same time.

CHAPTER 7

# RESULTS AND ANALYSIS

## 7.1 INTRODUCTION

In the previous chapter, we explained the various experiments that have been performed as part of this work. In this chapter, we aim to summarise the results collected from the experiments, present the findings, and explain the outcome of such findings.

### 7.1.1 Experiments - Challenges Involved

When performing the experiments undertaken in this work, the aim was to be able to perform the experiments in a controlled environment. This requires, for a given topology being tested, the presence of a wireless testbed that maintains consistent wireless connections between its hosts. The main objective is that the testbed behaves in the most consistent manner possible, for the different tests being executed.

In chapter 6, the different wireless topologies being tested were described. When setting up such topologies, a challenge exists in establishing stable wireless links between the hosts, in such a manner that the collective links reflect the desired topology. To obtain such topologies, the following procedures had to be undertaken:

1. Position the wireless nodes to represent the topology that is desired. Figure 7.1 represents the initial setup of the grid topology (topology 3) that was tested.
2. Adjust/reduce the transmit power of the wireless cards to prevent all nodes from hearing each other's wireless transmissions. The aim is to allow nodes to only communicate with immediate neighbours, hence creating a multi-hop environment amongst nodes that are out of wireless range from one another.
3. Tweak the positions of the wireless nodes in an attempt to obtain stable connections amongst neighbouring nodes.

Figure 7.1: Grid Topology - Topology 3.

Barring the grid topology (topology 3), we managed to obtain stable links that represented each of the desired topologies that were tested. Establishing a perfect grid topology proved too difficult due to the number of active wireless nodes situated in such a small proximity. Figure 7.2 represents the connection graphs of the different topologies tested, a double-headed arrow represents a stable link/connection between a pair of nodes.

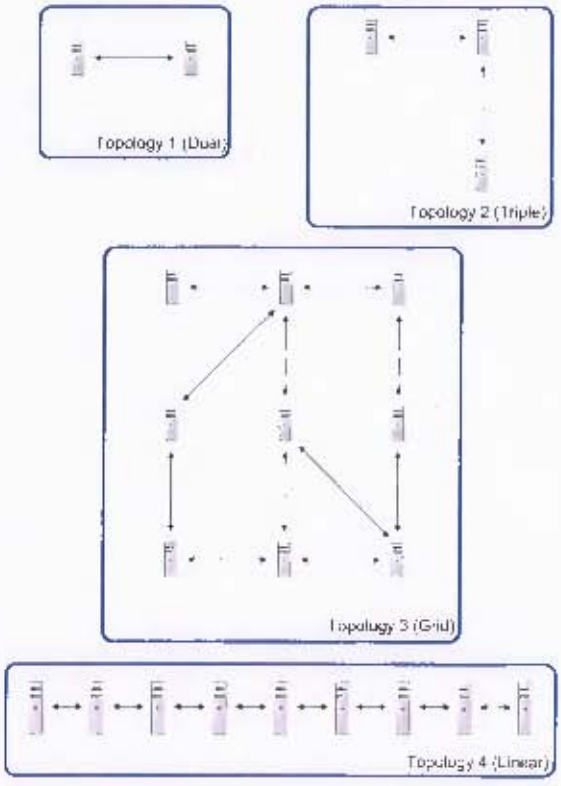



Figure 7.2: Established wireless link connections of the topologies tested.

Even though, for a given topology, the testbed was kept constant for all the different tests being performed, it must be stated that the wireless link quality, between pairs of nodes, did not always remain perfectly consistent over all the tests. Whilst these variations in link quality did occur, the overall topologies formed remained fairly stable, and the tests were run more than once in an attempt to obtain consistent results.

### 7.1.2 Experiments - Testing Samples

In order to display a summary of results, regarding a particular property being tested, one needs enough samples to ensure the summary to be consistent and reliable. It was observed in some of the tests, regarding certain properties being tested, that not enough samples had been gathered to allow any conclusive summary to be constructed, and hence displayed. Even though the test durations were more than long enough for routes to establish and stabilise, it was observed that in certain OLSR variations using the HORS signature scheme, sufficient samples of certain properties being tested could not be collected, due to the poor performance of the secure OLSR variation in question. In the results section, we have summarised the results in such a way that only properties being tested, for which sufficient samples have been captured, have been included.

## 7.2 Signature Scheme Tests

### 7.2.1 Signature Generation Times

Table 7.1 shows the elapsed time spent in the CPU for signature generation of the various signature schemes tested. The times represent an average taken over 3 test runs; for each test, elapsed times were obtained for 1000 iterations of each operation.

| | HORS 1024 | HORS 256 | IBS | RSA |
|---|---|---|---|---|
| Time (seconds) | 0.1032 | 0.1135 | 8.98 | 44.6167 |

Table 7.1: Signature Generation Times - 1000 iterations

From table 7.1, it is evident that the signature generation procedure of the HORS one-time signature scheme is substantially faster, and more efficient, than that of the IBS and RSA signature schemes tested. The results above confirm the efficiency of the HORS signature scheme, and other one-time signature schemes of a similar nature.

### 7.2.2 Signature Verification Times

Figure 7.3 shows the elapsed time spent in the CPU for the verification procedure of each signature scheme tested. Take note that the times represented are an average taken over 3 test runs; for each test, elapsed times were obtained for 1000 iterations of each operation.

For the HORS signature scheme, as more private keys from a key chain are used to create signatures, one can see how the verification times of those signatures increase linearly. This is due to the increasing number of hash function calls required, for verification purposes, of the signatures created by subsequent private keys in a key hash chain. The IBS and RSA verification

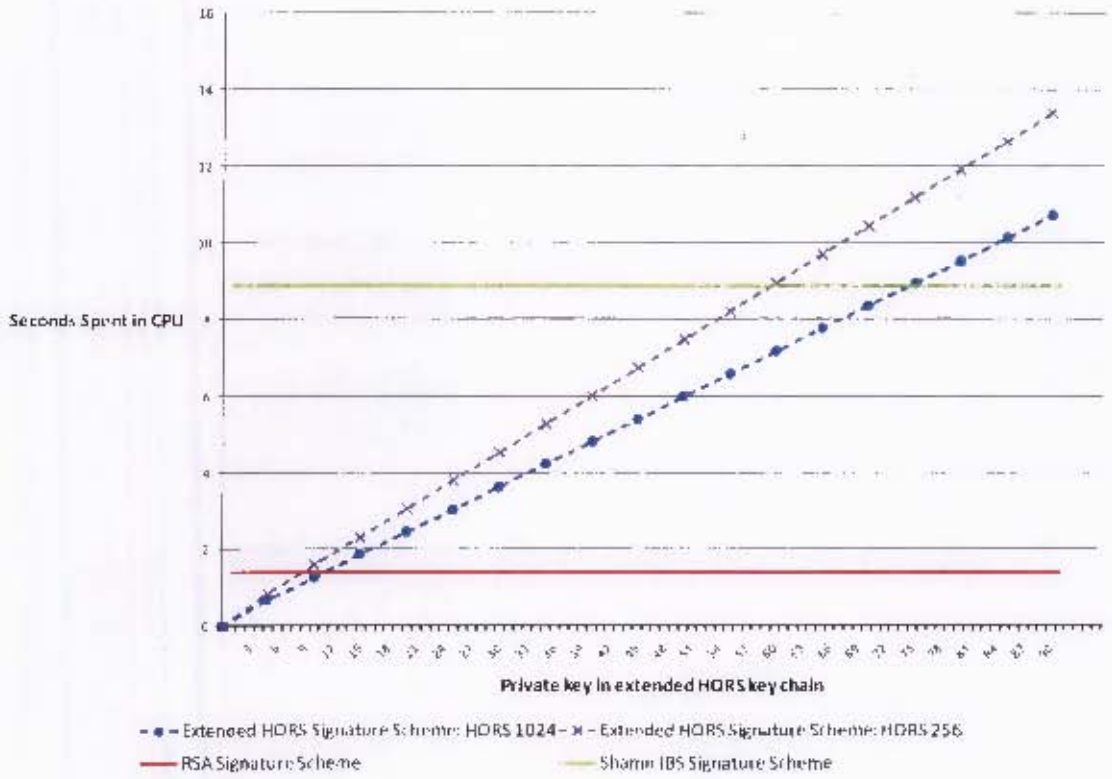



Figure 7.3: Signature scheme verification times.

times have been included in the graph for comparison reasons. The average elapsed time for the IBS and RSA verifications is constant as the two schemes are independent of any HORS parameters. If one considers the standard HORS signature scheme, with only a single private key in a key pair, it can be see that the HORS signature scheme is substantially faster, in terms of signature verification, than that of the IBS and RSA schemes tested. For the actual figures of the signature scheme verification results, refer to appendix B.

## 7.3 OLSR Message Reception

By comparing the sequence numbers of OLSR messages, originating from a given node, to those received by a given node, one can establish the percentage of OLSR messages that have been successfully received over a given hop-count. In our testing environment, there are several main factors that can affect the successful transmission of an OLSR message over more than a single hop, these are:

- Collisions of 802.11 Data Frames: It may be the case where two nodes, lying out of wireless range from one another, transmit data frames simultaneously. As these nodes exist out of range from each other, it is not possible for either of them to sense the transmission of the other. Hence, this may cause collisions of data frames at an intermediate node lying between the two nodes. This type of scenario is common in a multi-hop ad hoc network environment, and is commonly known as the hidden node problem (refer to appendix A).

- MPR Signalling: For a given OLSR node, MPR signaling determines which neighbour node is responsible for forwarding the OLSR messages originating from the node. If the MPR signaling process fails, messages originating from a node will not be forwarded by any receiving neighbours, and hence, will not be received by any further nodes in the network.

- Security Requirements: Upon reception, OLSR messages not meeting the specified security requirements will be dropped by the recipient node, and hence, will not be forwarded.

The graph shown in figure 7.4 represents the percentage of OLSR messages that were not received successfully over a varying hop-count. The figures represented in the graph are taken from the tests of topology 4, as this topology allows the greatest number of hops to be considered. It should be noted that the results represented in figure 7.4 are for a worst case scenario, due to the linear arrangement of the nodes. Any message that is not received successfully, by a single node in the linear channel, will not be forwarded to any further nodes, causing the message loss to be compounded as the hop-count increases.

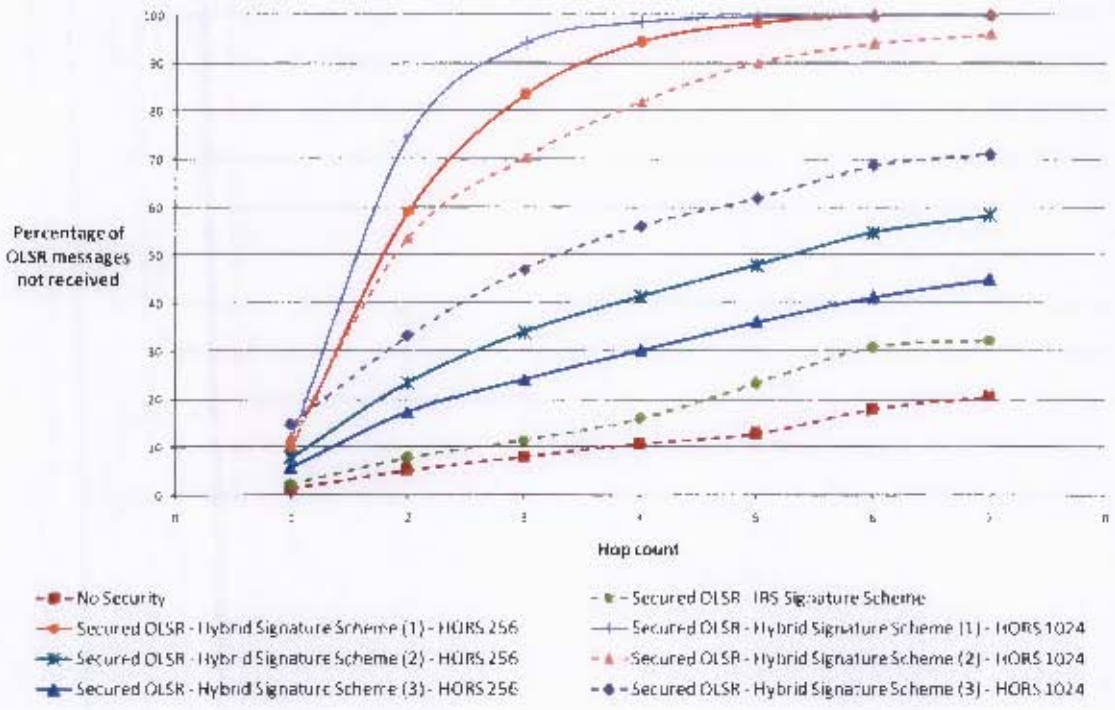



Figure 7.4: Percentage of OLSR messages not received over a varying linear hop-count.

Whilst the process of MPR signaling and that of enforcing security requirements affects the forwarding of an OLSR message at an intermediate node, these processes do not play part in the successful transmission of an OLSR message over a single hop, from an originator to an immediate neighbour. This is an important property to consider when looking at the OLSR message loss over a single hop.

| OLSR Variation | OLSR Messages Not Received (%) |
|---|---|
| No Security | 0.414 |
| Secured OLSR: IBS Signature Scheme | 0.718 |
| Secured OLSR: Hybrid Signature Scheme (1): HORS 256 | 0.482 |
| Secured OLSR: Hybrid Signature Scheme (1): HORS 1024 | 0.808 |
| Secured OLSR: Hybrid Signature Scheme (2): HORS 256 | 0.594 |
| Secured OLSR: Hybrid Signature Scheme (2): HORS 1024 | 0.416 |
| Secured OLSR: Hybrid Signature Scheme (3): HORS 256 | 0.737 |
| Secured OLSR: Hybrid Signature Scheme (3): HORS 1024 | 0.573 |

Table 7.2: Percentage of OLSR messages not received over a single hop - Topology 1.

Table 7.2 shows the percentage of OLSR messages not received over a single hop, taken from topology 1. Topology 1 represents only 2 OLSR nodes within wireless range of each other, hence, no other nodes are present to introduce potential frame collisions. If one compares the figures, over a single hop, of figure 7.4 (topology 4) to that of table 7.2 (topology 1), one can see how the introduction of additional nodes affects the reception of OLSR messages, for the different OLSR variations tested. For topology 1, the message loss percentage is very small, and similar for all the OLSR variations. Whilst the topology 4 graph shows the loss percentage, over a single hop, to be higher than that of topology 1 for all OLSR variations, it is interesting to note how the OLSR variations using HORS suffer significantly worse with the introduction of additional nodes. The reason for this is most likely due to the higher traffic overhead resulting from the large OLSR messages required for HORS public key distributions. With a large frame size, on average, for the OLSR variations using HORS, collisions are more likely to occur amongst wireless hidden nodes in the network. The loss of OLSR messages affects the HORS OLSR variations most severely, as it makes the process of public key distribution lengthly, and expensive from a traffic overhead point of view. The traffic overhead produced by the different OLSR variations is discussed in the following section.

## 7.4 OLSR Traffic Overhead

The OLSR traffic test monitored the traffic overhead produced by the OLSR variations tested. The addition of security mechanisms to the OLSR protocol introduces additional traffic overhead in terms of signatures and added security data to each message. Figure 7.5 presents a summary of the overhead, in size, of an OLSR message, secured using the security mechanisms discussed thus far in this work.

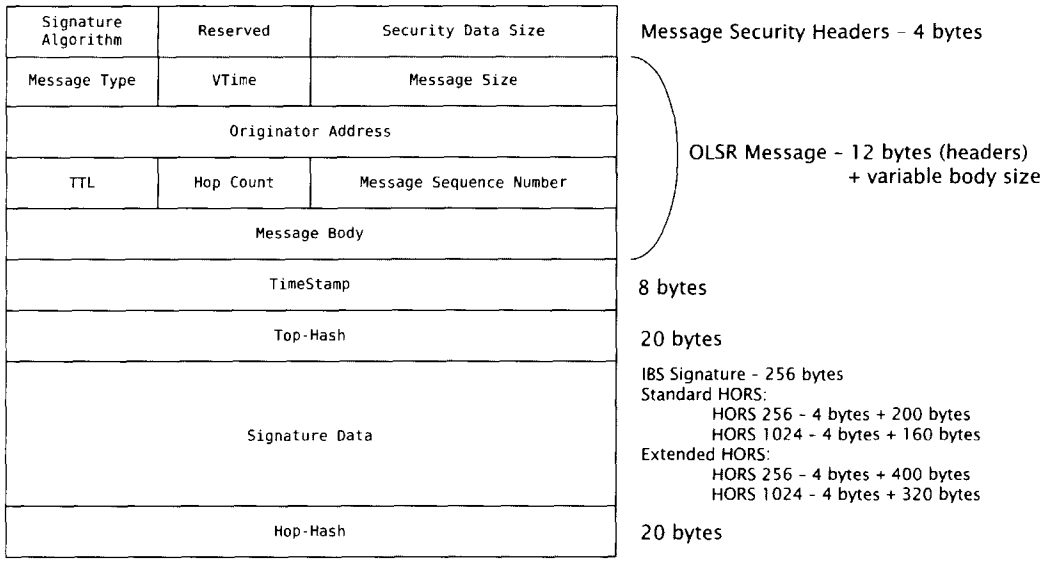



Figure 7.5: OLSR message size overhead with Security mechanisms

The traffic overhead graphs represented in figure 7.6 have been constructed from the $4^{th}$ topology tested (straight line topology of 9 nodes). The traffic was captured on the central node of the line topology, as a large portion of the OLSR traffic traverses across this node. Figure 7.6 shows 4 graphs, each representing traffic overhead figures for two different OLSR variations that were tested. The X-axis of each graph represents a time-line of the duration of the tests; the Y-axis of each graph represents a data rate, in kilobits per second (kbps), at a given time. Each data point, present in each graph, represents a data rate value, in kbps, averaged over a 10 second window period. Lines have been drawn through the averaged values to allow trends, if any, to be seen.

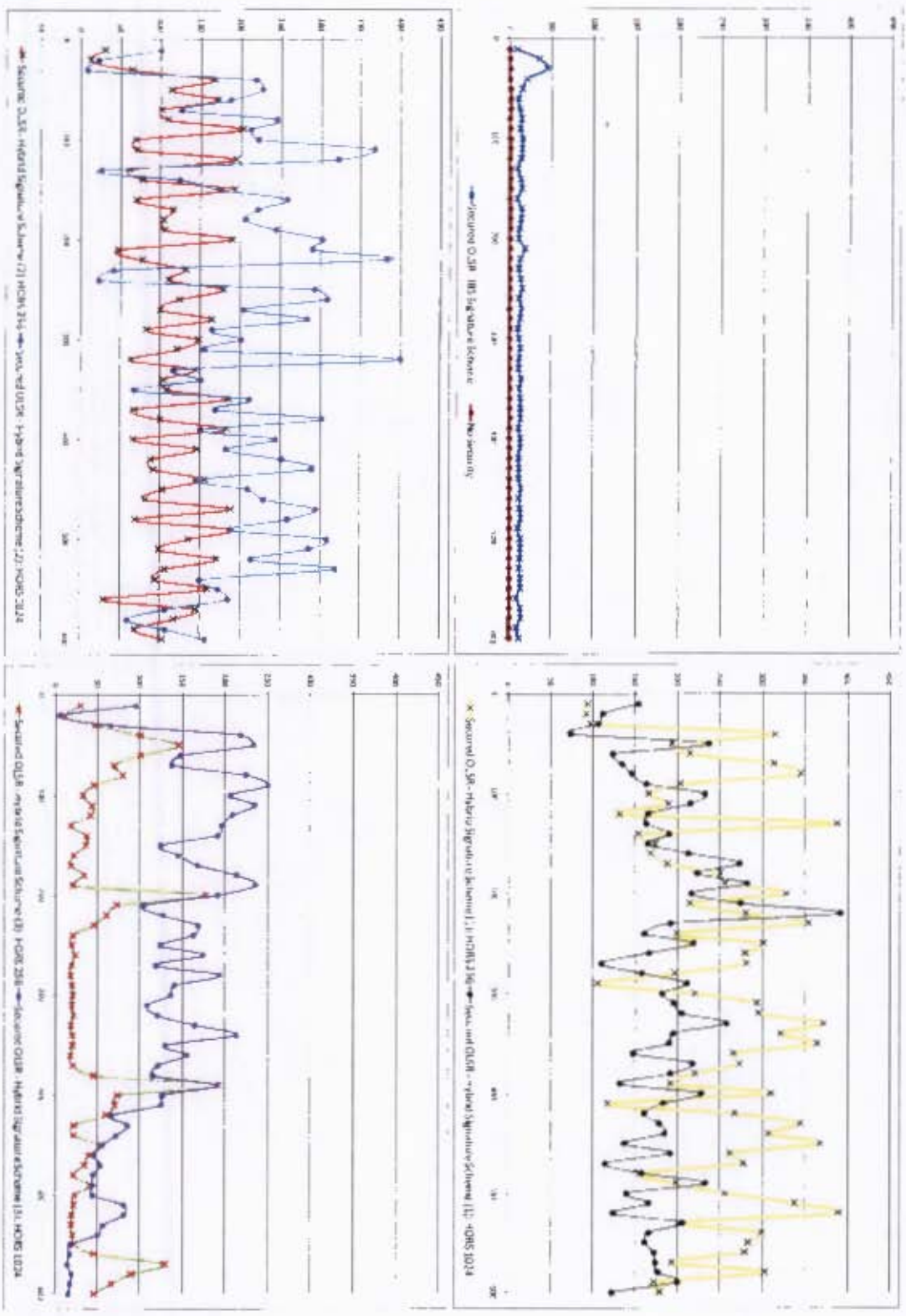


Figure 7.6: Control traffic overhead graphs of the tested OLSR variations

One can immediately see that the addition of security to the OLSR protocol adds a significant amount of traffic overhead. Amongst all of the security variations tested, the IBS signature scheme performs the best in this department. The HORS signature schemes suffer badly from the overhead incurred by public key distributions, resulting in random and inconsistent traffic patterns. Generally, the HORS variations with the smaller key chains suffer worst; the requirement of regular key refreshes, and the subsequent distribution of public keys, generates large amounts of traffic overhead. This is compounded by the loss of OLSR packets on a hop-by-hop basis - as shown in the previous section - as the loss of any OLSR messages, carrying a HORS public key, may lead to further public key distributions being required, creating a further increase in traffic overhead.

The standard HORS variation (variation 1) incurred the most overhead, the lifespan of a key pair under the standard HORS scheme is very short, producing an excessive amount of traffic overhead due to public key distributions. The HORS variation utilising the largest key chains (variation 3), performed significantly better than the other HORS variations. This is of no surprise, as the long lifespan of a key chain mitigates the public key distribution requirements. Table 7.3 shows the average data rates, in kbps, of the different OLSR variations represented in the graphs above.

| OLSR Variation | Average Data Rate (kbps) |
|---|---|
| No Security | 1.9444 |
| IBS Signature Scheme | 14.8138 |
| Hybrid Signature Scheme (1) - HORS 256 | 246.2702 |
| Hybrid Signature Scheme (1) - HORS 1024 | 182.4902 |
| Hybrid Signature Scheme (2) - HORS 256 | 114.4264 |
| Hybrid Signature Scheme (2) - HORS 1024 | 202.5282 |
| Hybrid Signature Scheme (3) - HORS 256 | 45.8566 |
| Hybrid Signature Scheme (3) - HORS 1024 | 123.9047 |

Table 7.3: Traffic Overhead of the OLSR Variations - Average Data Rate.

## 7.5 ROUTE DELAYS

### 7.5.1 Route Establishment

The route establishment test seeks to determine the amount of time, on average, the different OLSR variations require to establish routes to other nodes of different hop-counts. The security mechanisms present in the secure OLSR variations impact the process of establishing routes quite significantly, particularly as the hop-count grows. The overhead, in terms of establishing routes, is brought on primarily by the process of exchanging security information between communicating nodes, namely:

- Loose-time synchronisation exchange: Prior to a node accepting OLSR routing control messages, from a given sender, it must have performed a loose time synchronisation with the message originator. This round-trip exchange adds additional time to the route establishment process, particularly if one of the exchanges is not received correctly by one of the parties, in which case the loose time synchronisation process will have to be repeated.
- HORS public key exchange: This applies only to OLSR variations utilising the HORS signature scheme. This property of the HORS signature scheme adds significantly to the route establishment process. A recipient node cannot verify the HORS signature, corresponding to a received OLSR message, unless it is in possession of the HORS public key belonging to the message originator. The process of exchanging a HORS public key can be time consuming, considering the size of HORS public keys, and the possibility of packet loss during the exchange.

Figure 7.7 shows the average route establishment times of the OLSR variations, taken from topology 3 (grid). Figure 7.8 shows the average route establishment times taken from topology 4 (linear).

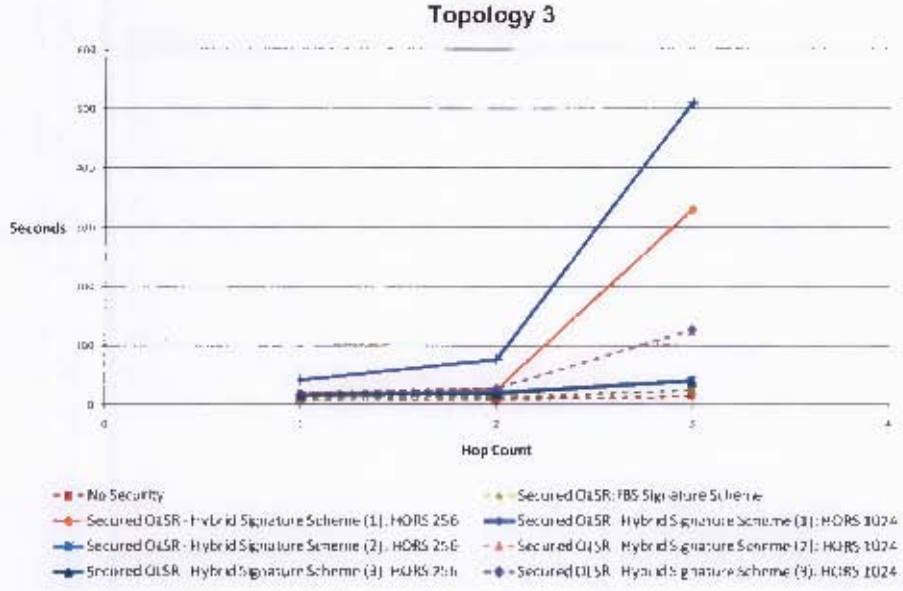



Figure 7.7: Average route establishment times of the tested OLSR variations - topology 3 (grid)

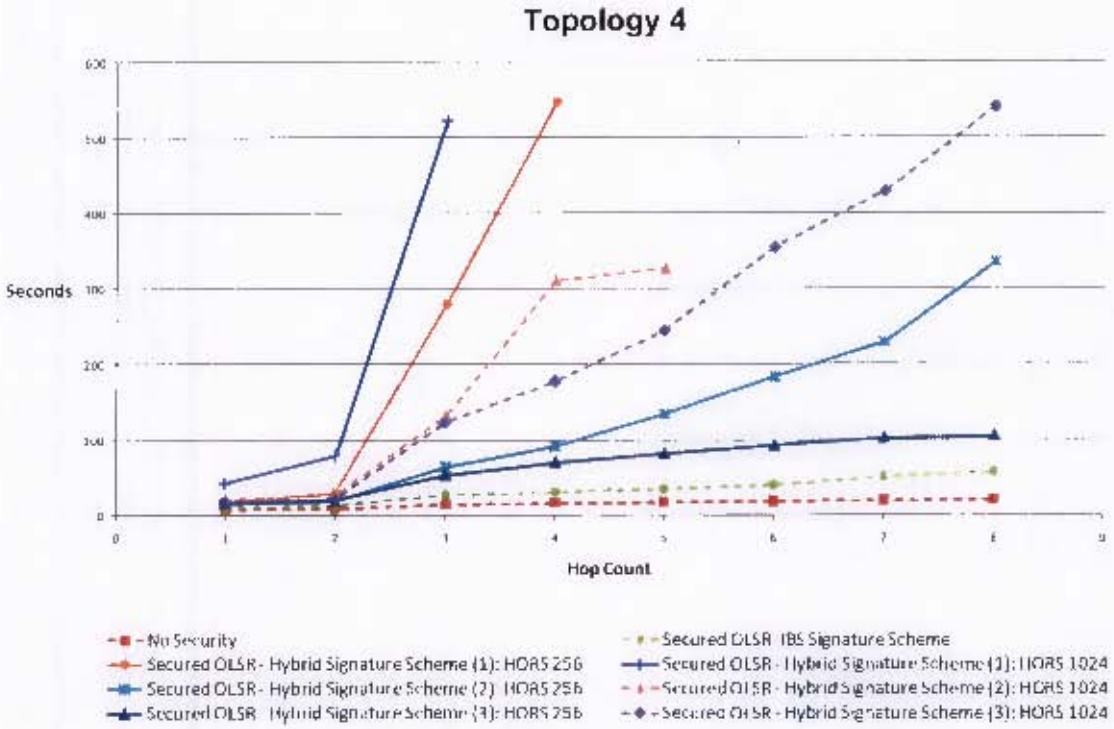



Figure 7.8: Average route establishment times of the tested OLSR variations - topology 4 (linear)

At low hop-counts, 1-2 hops, and even 3 for some secure OLSR variations, the secure OLSR protocols perform reasonably well compared to the standard OLSR protocol. However, at higher hop-counts, the security requirements of the secure OLSR protocols begin to impact significantly on the performance of OLSR. Once again, the OLSR variations utilising the HORS signature scheme suffer the most. The HORS schemes with longer key chains perform best, this is due to the long lifespan of a single HORS public key. The OLSR variations utilising the standard HORS scheme perform the worst, this is due to the requirement of regular key refreshes.

As mentioned already, the loss of OLSR packets during transmission makes the process of distributing a HORS public key, successfully, more difficult. OLSR packet loss, concerning the transmission of a HORS public key between any two given nodes, will most likely result in further public key requests taking place, and the subsequent distribution of a HORS public key. The problem with this scenario is that if the lifespan of a HORS public key is short, as is the case with the standard HORS scheme, by the time the current HORS public key of a node A is received successfully by a node B, node A may very well have performed, or soon perform, a key refresh. This would require node B to repeat the process of obtaining the new/current public key of A, and hence make the route establishment time between node B and A even longer. At higher hop-counts, the OLSR variations utilising the HORS signature scheme (standard or extended scheme) simply become infeasible, as the time taken to establish routes, over the higher hop-counts, grows too large relative to the standard OLSR protocol.

The loose time synchronisation exchange also adds quite significantly to the time in establishing a route. Consider the OLSR variation using the IBS signature scheme as an example: at low hop-counts, the difference between the secure IBS-OLSR variation and the standard OLSR protocol is quite small. However, just from the requirement of loose time synchronisation, the performance difference tends to grow rapidly as the hop-count increases, with the IBS-OLSR variation taking over twice as long to establish a route to a node 8 hops away.

### 7.5.2 **Route Repair**

The route repair test aims to determine how long, on average, it takes to repair a route of a given hop-count, if the route becomes unavailable. In our testing circumstances, there are several factors that can lead to a route becoming unavailable, these are:

- OLSR Packet Loss: Consider a given node, node A, maintaining a route to another node, node B. Node A's route to B can become unavailable due to an insufficient amount of OLSR control messages, that advertise a route to B, being received by node A. The route to B, from node A's point of view, can only repair once node A receives OLSR control messages advertising a route to B.
- HORS Public Key Refresh: A HORS key refresh leads to a subsequent distribution of the new HORS public key. For a given node A distributing a new HORS public key, the delay involved in distributing the public key successfully, to all intended recipient nodes, may lead to the recipient nodes being unable to verify OLSR control messages sent by A, for the duration of the delay incurred by the public key distribution. For recipient nodes, not in possession of A's current/new public key, this may lead to a loss of the routes advertised by node A.
- Security Requirements: The failure of a security check, such as a timestamp verification, on a secure OLSR message that has been received, will lead to the message being dropped, and hence, not processed or forwarded. If a significant amount of OLSR control messages are dropped due to security checks, this may lead to a loss of the routes being advertised in the OLSR messages being dropped.

Figure 7.9 shows the average route repair times of the OLSR variations, taken from topology 3 (grid). Figure 7.10 shows the average route repair times taken from topology 4 (linear). Regarding the average repair times for the different OLSR variations tested, not much elaboration is required as the repair times follow a similar pattern to that of the route establishment times. Once again, the OLSR variations utilising the standard HORS signature scheme suffer the most due to their dependency on regular key refreshes.

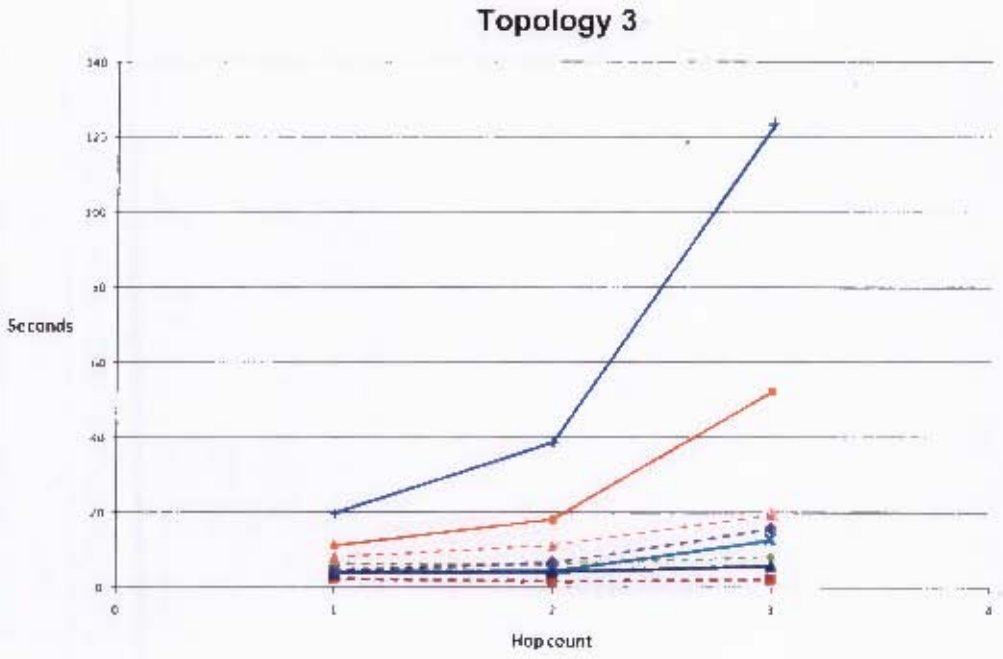



Figure 7.9: Average route repair times of the tested OLSR variations - topology 3 (grid)

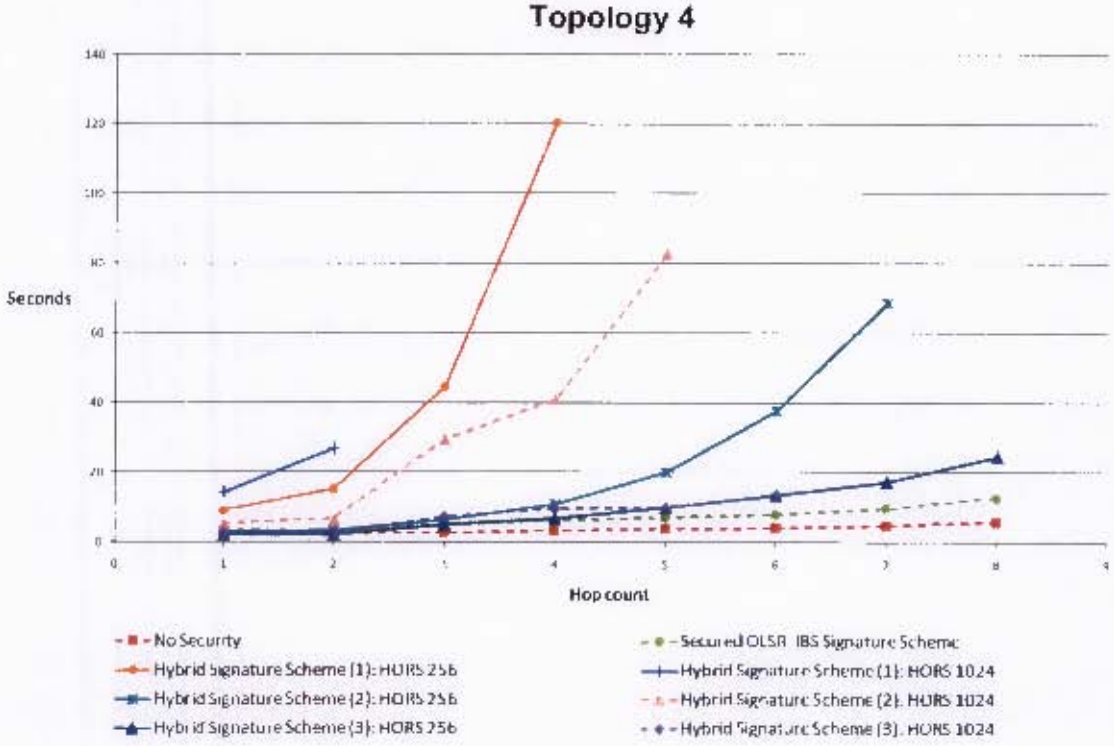



Figure 7.10: Average route repair times of the tested OLSR variations - topology 4 (linear)

## 7.6 ROUTE AVAILABILITY

The route availability test aims to determine a how available a route is, as a percentage, for the duration of the test. The test measures the availability of a route only once the route has been established. The factors that can affect route availability have been discussed in the previous section, and hence will not be discussed again in this section. Figure 7.11 represents the availability of routes of the different OLSR variations tested from topology 4 (linear).

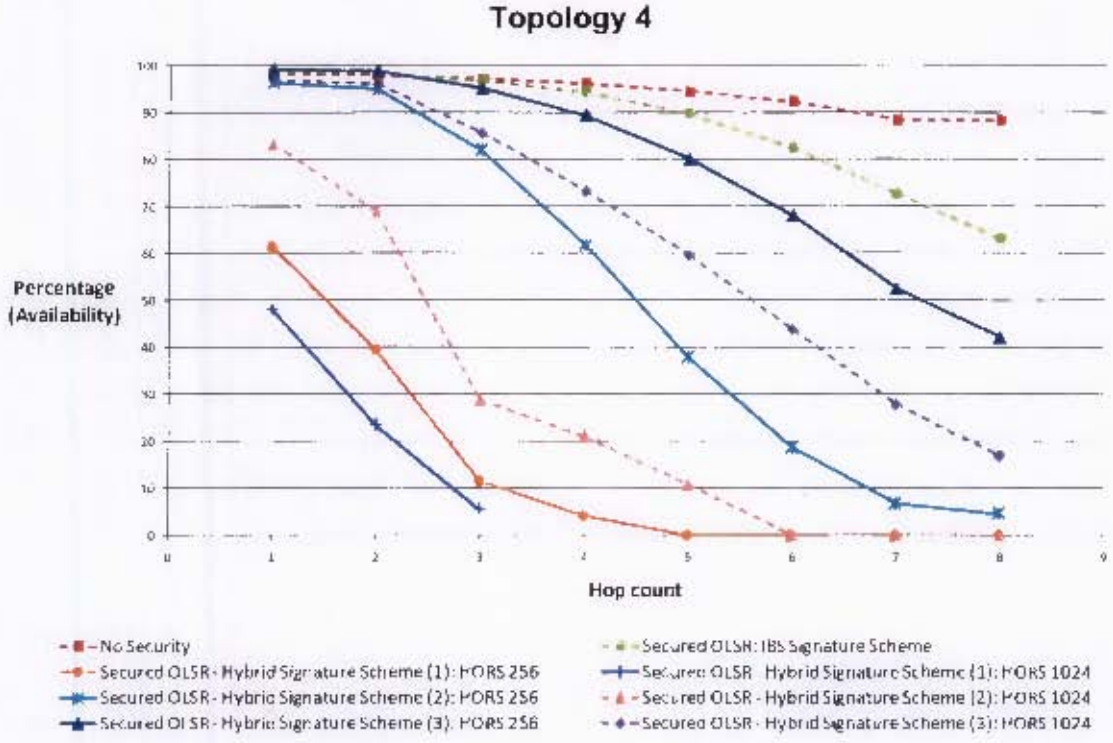



Figure 7.11: Route Availability - Topology 4 (linear)

Relative to the standard OLSR protocol, the secure OLSR variations perform badly as the hop-count tends to grow. The OLSR variations utilising the HORS signature scheme perform worst. The route availability of these OLSR variations is effected most by:

- HORS Key Refreshes: As already discussed previously, the difficulty associated with a HORS public key distribution can lead to recipient nodes being unable to verify OLSR messages signed by a corresponding HORS private key. The greater the dependency on HORS key refreshes, the greater the possibility that the difficulty associated with a HORS public key distribution will affect the protocol in a negative manner.
- OLSR Packet Loss: It appears that our OLSR variations utilising HORS suffer most from collisions on the wireless medium. This was shown in figure 7.4. where the HORS OLSR variations suffered most in terms of OLSR message reception. An excessive amount of packet loss will lead to routes becoming lost or unavailable, particularly as the hop-count increases together with the likelihood of packet loss.

- Failed Timestamp Verifications: When an timestamp verification fails, the OLSR message is dropped, and hence, not processed by the recipient node and not forwarded to any further neighbouring node. Thus, a significant amount of failed timestamp verifications can lead to routes becoming unavailable, as mentioned in the previous section.

The IBS-OLSR variation performs the closest to the standard OLSR protocol; however, due to the loss of OLSR messages from failed timestamp verifications, and an increased likelihood of collisions on the wireless medium due to larger frame sizes, the performance difference of the IBS secure variation tends to drop off significantly at higher hop-counts. If one could have ignored the loss of OLSR messages due to failed timestamp verifications, the IBS-OLSR variation would have actually performed relatively well, compared to the standard OLSR protocol. The next section goes into more detail regarding loose time synchronisation and timestamp verifications.

## 7.7 LOOSE TIME SYNCHRONISATION

As has previously been mentioned, assuming time synchronisation amongst nodes in a spontaneous networking environment, such as an ad hoc network, is not a very realistic assumption. One of our goals was to test the use of a loose time synchronisation protocol, to enable the means of end-to-end timestamp verification of an OLSR message. When using the loose time synchronisation protocol over multiple hops, as in our case with OLSR, potential problems may arise:

- The loose time synchronisation protocol can be exploited by intermediate malicious nodes, as discussed in section 3.4.3, from chapter 3.
- Accuracy: Unexpected or inconsistent transmission, processing, and queuing delays at each hop can render the loose time synchronisation inaccurate. This inaccuracy can lead to inaccurate timestamp verifications, and hence, cause poor performance of the secure OLSR protocols.

One of the first aspects looked at was the number of OLSR messages dropped due to the timestamp verification of OLSR messages. Table 7.4 shows the number of OLSR messages dropped as a percentage of the total number of timestamp verification samples taken. The table summarises the results taken from topologies 3 (grid) and 4 (linear) of all the secure OLSR variations, categorized over different hop-counts.

| Hop Count | IBS | HSS: HORS 256 | | | HSS: HORS 1024 | | |
|---|---|---|---|---|---|---|---|
| | | (1) | (2) | (3) | (1) | (2) | (3) |
| 1 | 0 | 2.59 | 0.05 | 0.003 | 12.02 | 6.13 | 2.58 |
| 2 | 0.18 | 10.26 | 0.41 | 0.33 | 36.97 | 20.26 | 7.36 |
| 3 | 3.15 | 9.76 | 1.35 | 2.9 | - | 12.21 | 5.72 |
| 4 | 5.97 | - | 2.42 | 2.58 | - | 9.87 | 4.79 |
| 5 | 5.06 | - | 2.97 | 2.67 | - | - | 8.03 |
| 6 | 4.25 | - | 2.13 | 2.56 | - | - | 5.17 |
| 7 | 6.12 | - | 1.36 | 3.85 | - | - | 5.88 |

Table 7.4: Percentage of unsuccessful timestamp verifications

* IBS → Secured OLSR: IBS Signature Scheme
* HSS → Secured OLSR: Hybrid Signature Scheme

Interpreting the results of the loose time synchronisation, and subsequent verifications, proved quite difficult. Already, from the table above, one can see that no consistent trends tend to exist as the hop-count increases. However, for most of the secure OLSR variations, the results do indicate that over a single hop the loose time synchronisation protocol, and subsequent timestamp verifications, perform fairly well, with relatively low percentages in dropped messages. Over more than a single hop however, the percentage of failed timestamp verifications tends to increase quite substantially.

Whilst figures of around 2% or 5%, for some of the secure OLSR variations, may seem small, the effect of messages being dropped due to failed timestamp verifications is compounded as the hop-count increases; for instance: given a node 7 hops away from a given sender, not only is the node dropping messages it receives due to failed timestamp verifications, it is also not receiving the OLSR messages being dropped by MPR nodes along the route of the message to the node in question.

With respect to the OLSR variations using the HORS signature scheme and timestamp verifications, the HORS 256 parameter set performs far better than that of the HORS 1024 parameter. The reason for this is most likely due to the higher and more inconsistent traffic overhead patterns of the HORS 1024 parameter set, causing additional processing, queueing and transmission delays at each hop. This is also evident when comparing the results of the OLSR variations using the standard HORS scheme, which generate higher traffic overhead, to the OLSR variations using the extended HORS signature scheme. The OLSR variations using the standard HORS signature scheme tend to result in a far higher percentage of failed timestamp verifications.

A second aspect looked at was the resulting average slack of the timestamp verifications. In other words, the difference between the time reflected in a timestamp, and that of the comparison time calculated, based on a sender's upper-bound time. Table 7.5 reflects the average slack

(in seconds) of timestamps verifications, summarised from the results of the tests performed on topologies 3 (grid) and 4 (linear).

| Hop Count | IBS | HSS: HORS 256 | | | HSS: HORS 1024 | | |
|---|---|---|---|---|---|---|---|
| | | (1) | (2) | (3) | (1) | (2) | (3) |
| 1 | 0.588 | 0.631 | 0.539 | 0.561 | 0.609 | 0.547 | 0.525 |
| 2 | 0.538 | 0.663 | 0.657 | 0.578 | 0.373 | 0.408 | 0.576 |
| 3 | 0.593 | 0.653 | 0.749 | 0.534 | - | 0.640 | 0.658 |
| 4 | 0.498 | - | 0.784 | 0.726 | - | 0.697 | 0.807 |
| 5 | 0.575 | - | 0.912 | 0.830 | - | - | 0.658 |
| 6 | 0.670 | - | 1.235 | 0.872 | - | - | 0.872 |
| 7 | 0.643 | - | 1.377 | 0.858 | - | - | 1.051 |

Table 7.5: Timestamp verifications - average slack

* IBS → Secured OLSR: IBS Signature Scheme
* HSS → Secured OLSR: Hybrid Signature Scheme

Considering that 0.5 seconds of slack had already been introduced into the timestamp verification calculation, as mentioned in section 6.6, the timestamp verifications on average proved fairly accurate. One can essentially subtract 0.5 from the above figures to get an idea of the real average slack, i.e. the average difference had our own slack not been introduced. Once again, over a single hop the timestamp verification proves, in general, the most accurate. As the hop-count increases, the general trend of the slack difference tends to increase.

The problem that exists with the loose time synchronisation, over multiple hops, is that at the time of the synchronisation, it is a once-off request/response exchange. Whilst this exchange between any two given nodes may be accurate in some instances, in other cases it may not represent the true round-trip-time at all times of the communications between the two nodes. Even if averages are worked out from subsequent loose synchronisation exchanges, as we did in our implementations, it is difficult to accommodate all the possible delays associated with the transmissions of different messages, whilst attempting to keep the resulting slack as low as possible.

Consider the IBS-OLSR variation as an example. The IBS-OLSR variation proved most accurate, with the most consistent average slack. This consistency is most likely due to the fact that, out of all the secure OLSR variations, the IBS-OLSR produces the most regular and consistent traffic patterns, and hence, fairly consistent delays at each hop. The IBS-OLSR protocol however, barring the HORS 1024 OLSR variations , also suffered the most in terms of failed timestamp verifications at higher hop-counts. This shows that even whilst, on average, the IBS-OLSR variation resulted in fairly accurate loose time synchronisations, it still resulted in too many messages falling out of the allowed timestamp window at higher hop-counts. Regarding the OLSR variations using the different HORS schemes, the average slack tends to increase

quite significantly as the hop-count increases. This increase is most likely due to inaccuracies of the RTTs calculated from the loose time synchronisation exchange. This once again reinforces the difficulty involved in establishing a round-trip-time, over multiple hops, which accommodates all the potential delays of an incoming message, particularly in an environment of inconsistent and high traffic volumes.

## 7.8 VALIDITY WINDOW - EXTENDED HORS PRIVATE KEYS

In section 3.7.2, the use of a window period for private keys in an extended HORS key chain was discussed. The goal of the window period is to prevent malicious nodes from forging signatures using derived values of old private keys. One goal of the testing was to determine the accuracy of the window period, and to determine if the window period is feasible. One aspect to look at is the remaining slack of the time period for each private key, i.e. the difference, in time, when receiving the $r^{th}$ and final message of a sender's current private key, to that of the calculated cut-off/upper-bound time of the window period for the sender's current private key.

In section 6.6, it was stated that 0.25 seconds of slack, per message, was introduced into the calculation of a private key time period. Considering the two different parameter sets, the following amount of slack was introduced for each:

- HORS 256: $slack = 0.25 \cdot r = 0.25 \cdot 2 = 0.5$ (seconds)
- HORS 1024: $slack = 0.25 \cdot r = 0.25 \cdot 6 = 1.5$ (seconds)

The slack figures shown above are then introduced into the calculation of the private key window period, as shown by the calculations in section 3.7.2, from chapter 3. Determining an amount of slack to add to the calculations was not trivial, the goal was to allow enough slack such that all $r$ messages, signed by a private key, could be received successfully without creating too much slack. The difficulty lies in predicting the various delays associated with the transmission of an OLSR message across a varying number of hops: one message may experience considerably more delays, over a number of hops, than that of another message. It was decided that a small amount of slack would be added for each of the $r$ messages such that, collectively, the slack would be enough to allow all $r$ messages, signed by a single private key, to be received successfully within the private key's window period.

Table 7.6 represents a summary of the average slack, of the private key time period, observed from the tests of topologies 3 and 4. The results have been taken from the tests of the extended HORS OLSR variations, and have been grouped into the two different HORS parameter sets (HORS 256 and HORS 1024). When calculating a valid time period for a HORS private key, $key_n$, the aim is to keep the time period as accurate as possible, with as little slack as possible. This is essentially to prevent neighbours from deriving values of $key_n$, from subsequent private keys in the same extended HORS key chain as $key_n$, and then forging signatures using $key_n$. The slack needs to be small enough such that for a given private key $key_n$, it is infeasible in the given time-frame for an adversary to 1) wait for messages signed by a subsequent private key, 2) derive values of $key_n$ from the signatures of the subsequent private key, and 3) forge a signature of a message using $key_n$, and ensure the reception of the forged message at the intended victim.

| Hop Count | Extended HORS 256 | Extended HORS 1024 |
|---|---|---|
| 1 | 1.225 | 3.247 |
| 2 | 2.684 | 5.193 |
| 3 | 2.989 | 5.941 |
| 4 | 3.128 | 6.064 |
| 5 | 3.329 | 6.976 |
| 6 | 3.397 | 5.960 |
| 7 | 3.315 | 6.504 |

Table 7.6: HORS Private Key Window Period - Resulting Slack

Considering the amount of slack that was introduced, the accuracy of the valid time period for an extended HORS private key, over a single hop, is fairly good. Over a greater hop-count however, the slack grows substantially. The reason for this could be threefold:

- Slack Introduced: It is possible that by introducing an amount of slack for each message that may be signed by a HORS private key, the total slack for a given HORS private key may have amounted to an excessive figure.
- Carried Over Slack: As we have shown, the window period calculation for a given HORS private key, in an extended key chain, may result in the calculation leaving a certain amount of excess slack. Owing to the fact that the cut-off point of a window period for a HORS private key is used as the starting time for the window period of the next private key, of the same key chain, it is possible for any excess slack to carry over to the time window calculation of the subsequent private key.

  In section 3.7.2, we described a possible approach to minimise the carried over slack of a previous private key window. By having calculated a timestamp at the time of receiving the most recently signed message, of the previous private key belonging to a given neighbour, the timestamp can be used as a starting point for a window period of the subsequent private key, of the neighbour's same key chain. However, the problem with this approach, concerning the window period calculation of a current private key of a neighbour, is that it is most effective if the starting point timestamp had been calculated at the reception time of the last, or $r^{th}$, message signed by the neighbour's previous private key. Hence, the more recent the last successfully received OLSR message, of the messages signed by a neighbours previous private key, the more effectively slack can be reduced for the window period of the next private key. Poor OLSR message reception can therefore negate, to a large extent, this approach of minimising excess slack from the window period of a previous HORS private key in an extended HORS key chain.

- Window Period Calculation: Our calculation involved the use of the OLSR HELLO message interval time to calculate an upper-bound window period for incoming messages, of a given neighbour, that have been signed by the current HORS private key of the neighbour. The calculation did not incorporate the interval times of other possible messages: in the case of our experiments, the Topology Control (TC) message. In hindsight, and taking the previous two points into consideration, the use of the TC message interval time would have made the window period calculation more accurate, leaving less excess time open for exploitation.

The excess slack times of each private key window period is too large for practical use. By having such a large slack period, the extended HORS scheme can be exploited by attackers forging signatures for old private keys, as we have previously shown. The experiments on the indoor testbed allowed the problems of our private key window period to be noticed, whilst potential solutions to such problems, as an attempt to make the window period more accurate, have been given above.

## 7.9 PING TEST

The ping round-trip times (RTTs) give a general reflection of the delays in transmitting and processing an OLSR control message across a number of hops. Under the standard OLSR and IBS-OLSR variations, enough samples of round-trip-time ping messages were able to be collected to allow suitable averages, for varying hop-counts, to be calculated. However, the method used to transmit and receive OLSR ping messages did not work very well under the HORS OLSR variations, for the following reasons:

1. MPR Signalling: Under the standard OLSR protocol in our experiments, only MPR nodes generate messages with maximum Time-To-Live (TTL) values, these being Topology Control messages. Hence, when it came to sending custom OLSR ping request/response messages, it was decided that only MPR nodes should set the TTL value to maximum, whereas non-MPR nodes should set the TTL value to 1. This was done to prevent unnecessary key and time synchronisation exchanges taking place amongst ping nodes who were not MPR nodes - note however, for the standard OLSR protocol where no security exchanges take place, the TTL of all ping messages was set to its maximum value for all nodes. Under certain OLSR variations using HORS, the MPR signaling amongst neighbours performed relatively badly due to lost/dropped OLSR messages. The MPR signaling process therefore affects the transmission range of the ping messages, and hence, can make it difficult to obtain a sufficient amount of multi-hop ping samples if the signaling process does not perform effectively.

2. Packet Loss: If either of the ping request or response messages are lost during the exchange, the ping exchange becomes worthless and will have to be repeated. If there is a high loss rate of OLSR packets in a network, this can negatively affect the number of ping samples that can be obtained within a given timespan.

Figure 7.12 shows the average ping RTTs of several of the tested OLSR variations, for varying hop-counts. Only those variations for which enough samples could be collected have been included in the results. Whilst it proved difficult to obtain enough samples for all the hop-counts, the results in figure 7.12 still allowed certain conclusions to be drawn.

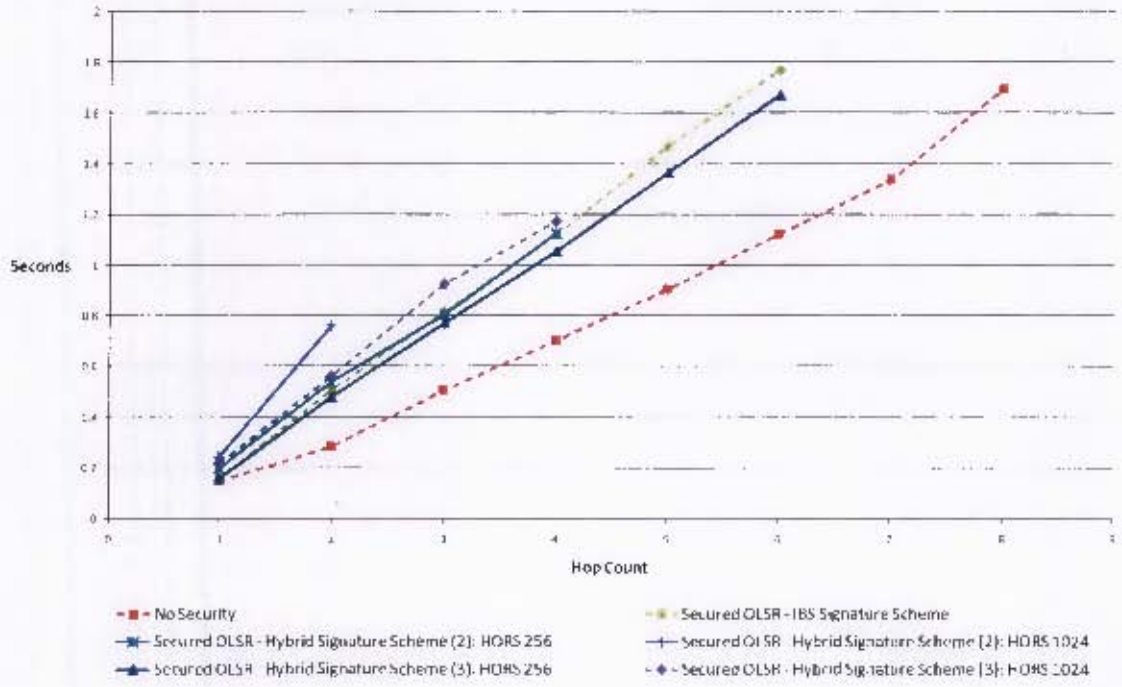



Figure 7.12: Round-Trip-Time of Custom OLSR Ping Message - Topology 4 (linear)

What is immediately clear, from the ping results, is that the addition of security mechanisms to the OLSR protocol adds a significant amount of delay at each hop. The ping times of all the secure OLSR variations are significantly higher than the ping times of the standard OLSR protocol, this is not surprising considering the delays involved with signature generation and verification at each hop. What is interesting to note, however, is the average time of the pings of the OLSR variations using HORS, relative to the ping times of the IBS-OLSR variation. From section 7.2, it was shown that the HORS, and extended HORS, signature schemes offered signature generation and verification operations significantly faster than that of the IBS signature scheme tested. However, out of all the OLSR variations using HORS, only OLSR using the extended HORS 256 scheme, with the longer of the key chains, produced ping times marginally lower than that of the IBS-OLSR variation. This extended HORS 256 OLSR variation also produced, out of all the OLSR variations using HORS, the lowest average traffic overhead, as shown in table 7.3. This once again confirms how significantly the higher traffic overhead, produced by the OLSR variations using HORS, affects the transmission delays of an OLSR message over a varying number of hops.

## 7.10 SUMMARY

In this chapter, the results obtained from the experiments have been summarised and presented. The findings regarding each test have been discussed with the initial objectives of the research in mind. Whilst interpreting certain aspects of the results was not easy, logical explanations for the outcomes of each test were always sought. The next chapter concludes this dissertation by presenting final conclusions and discussing potential areas of research with area for improvement.

CHAPTER 8

# CONCLUSIONS AND FUTURE WORKS

## 8.1 INTRODUCTION

In this chapter, we aim to 1) revisit the initial objectives of the research undertaken, 2) provide a brief summary of the results, 3) assess the results, with the initial objectives of the research in mind, and 4) provide recommendations for future works.

## 8.2 RESEARCH OBJECTIVES

Routing protocols for wireless ad hoc networks is the research area forming the main subject of this work, with specific emphasis on the provision of end-to-end security mechanisms for the Optimized Link State Routing (OLSR) protocol. The main research objectives of this work, with respect to security and the OLSR protocol, were:

1. Identify the need for end-to-end security in the OLSR protocol.
2. Investigate the use of specific end-to-end security mechanisms in the OLSR protocol.
3. Investigate the use of a one-time signature scheme, HORS, as well as our extended version of HORS, as an end-to-end broadcast authentication mechanism for the OLSR protocol.
4. Determine the feasibility of using such security mechanisms to secure the OLSR protocol.

To satisfy the objectives mentioned above, a security module incorporating end-to-end security mechanisms was developed for an existing OLSR implementation. Experiments were performed on an indoor, 9 node, wireless ad hoc testbed. The results obtained from the experiments allowed us to observe the effects of the security mechanisms on the OLSR protocol, as well as to satisfy the research objectives of the dissertation.

## 8.3 THE OLSR ROUTING PROTOCOL - END-TO-END SECURITY

In chapter 2, the OLSR protocol was introduced and an emphasis fell on the need to secure the OLSR protocol. Like many routing protocols designed for wireless ad hoc networks, the OLSR protocol provides no security features for the routing control data it produces. For this reason, as mentioned in chapter 2, the OLSR protocol is vulnerable to several security threats. To provide end-to-end security for OLSR routing control data, as discussed in chapter 2, the emphasis must fall on securing OLSR messages. Unlike OLSR packets which simply act as hop-by-hop carrier mechanisms, OLSR routing messages are end-to-end, from source to every intended destination in the network.

Whilst there is no doubt that securing the OLSR protocol is important, providing security for a broadcast protocol, such as OLSR, in a manner that does not hamper the protocol's performance severely, is not trivial. Broadcast protocols designed for wireless networks are especially difficult to secure, considering the sensitive and volatile nature of the wireless medium they operate on. Frame collisions in 802.11 wireless networks have been discussed, briefly, with the hidden node problem described in appendix A. For unicast transmissions in 802.11 wireless networks, mechanisms such as Request-to-Send/Clear-to-Send (RTS/CTS) [21] exist as a partial solution for the hidden node problem, and MAC layer acknowledgments, for received 802.11 frames, are used to detect lost frames for retransmission.

For multicast/broadcast transmissions in an ad hoc network, however, such mechanisms are not in place to assist with 802.11 frame transmissions, resulting in a lack of reliability in 802.11 broadcast/multicast transmissions [69]. Broadcast protocols are therefore much more susceptible to collisions of frames due to hidden nodes in the network. Also, without the use of frame acknowledgments by receivers, a sender cannot detect a lost frame, meaning no retransmission of the frame will take place. Taking these properties of 802.11 broadcasting into account, one can see the challenge involved with securing a broadcast protocol, such as OLSR, in an ad hoc network, particularly if the secure protocol is sensitive to packet loss.

From a security design point-of-view, the concept of using the proposed end-to-end security mechanisms for OLSR were viable, to a certain extent. However, as the results from the experiments have indicated, the practical use of such security mechanisms is not feasible, when considering a broadcast protocol over an unreliable transmission system which is sensitive to packet loss. This illustrates the importance of experimentation when testing the feasibility of design concepts.

## 8.4 RESULTS - CONCLUSIONS

### 8.4.1 Loose Time Synchronisation & Timestamp Verifications

One of the goals of this work was to determine how a simple loose time synchronisation exchange would perform as a basis for end-to-end timestamp verifications. Overall, the timestamp verifications performed quite poorly. Over a single hop, for most of the OLSR variation tested, the loose time synchronisation and subsequent timestamp verifications were very accurate. However, as the hop-count between a given sender and receiver increased, the results indicated that the accuracy tended to fall off substantially.

As discussed in the previous chapter, the difficulty lies in establishing a loose time relationship, for any given hop-count, such that the failure rate of timestamp verifications is low and the precision of the loose time synchronisation is fairly accurate. The results from the OLSR variation using the IBS signature scheme is an example of this: of all the OLSR variations tested, the IBS variation produced the most consistent average slack (accuracy), yet suffered quite poorly from failed timestamp verifications at higher hop-counts.

Whilst the loose time synchronisation exchange and timestamp verification calculation, used as part of the security implementation, proved suitable for a single hop exchange, its performance over multiple hops degrades too significantly. The difficulties that exist over multiple hop loose time synchronisations, coupled with the fact that the loose time synchronisation exchange can be exploited, as shown in section 3.4.3, simply makes the use of our timestamp security mechanism impractical for verification purposes of multi-hop communications.

### 8.4.2 **Identity-Based Signature Schemes**

Identity-based signature schemes offer a significant advantage with regard to public key distributions. Instead of the need for each node to distribute its public key, any node can derive the public key of another node through the use of publicly known information about the node, such as an IP address. This property of identity-based signature schemes can provide benefits in terms of the overhead, in both delays and traffic, associated with public key distribution.

Shamir's identity-based signature scheme [56], based on RSA functions, was used as an identity-based signature scheme for the OLSR security design and implementation done as part of this work. As the results indicate in the previous chapter, out of all the secure OLSR variations tested, the IBS-OLSR secure variation performed the best overall. Even though some of the results reflect a poor performance, particularly at high hop-counts, from the IBS-OLSR implementation, these results have been exaggerated due to the bad performance of the timestamp verification mechanism, as explained in the previous section. The IBS-OLSR implementation showed a significant advantage over the HORS-OLSR variations, particularly in terms of route establishment times and traffic overhead.

Whilst the IBS scheme offers several advantages, the disadvantages cannot be ignored. In section 7.2, the signature verification and generation results reflect how inefficient and slow our IBS scheme implementation is, relative to the HORS one-time signature scheme implementations. The use of computationally expensive signature schemes may not always be practical, particularly on devices with minimal resources and processing capabilities, such as devices in sensor networks.

For the experiments undertaken as part of this work, it was assumed that the nodes received fixed IP addresses, and thus the identity-based private keys could be pre-shared on the nodes, solely for experimental purposes. However, this scenario is not applicable or practical for every situation - in section 3.1, it was mentioned that the security design was not for a general case. Consider figure 2.13, from chapter 2, a traditional identity-based signature system makes use of a third party to provide users with identity-based private keys and the master public key. Also, networks often use of a centralised DHCP server for dynamic IP address allocation of network users. From a practical point of view, the two points just mentioned pose potential problems

for the use of identity-based signatures in an ad hoc network, this being due to the lack of infrastructure present, as well as the decentralised and dynamic nature of an ad hoc network. Dynamic IP address allocation in wireless ad hoc networks is still, very much, an active area of research.

### 8.4.3 The HORS One-Time Signature Scheme

One of the main research objectives of this work was to investigate the use of a one-time signature scheme as an end-to-end broadcast authentication mechanism for the OLSR protocol. For this purpose, the HORS one-time signature scheme was chosen as part of our security design and implementation. One-time signature schemes, such as the HORS scheme, offer very fast and efficient signature operations, however, may also suffer from various limitations as explained in section 2.7.1. Our aim was to determine if the use of the HORS signature scheme, as an end-to-end signature scheme for OLSR, was feasible in light of the limitations the signature scheme faces.

#### *The Standard HORS Signature Scheme*

Two of the secure OLSR variations tested incorporated the standard HORS signature scheme as a main digital signature scheme. The two OLSR variations made use of different parameter sets, namely: the HORS 256 and HORS 1024 parameter sets, as we term them in section 2.7.1. The main limitation behind the standard HORS signature scheme is the limited number of signatures a given key pair can provide. This limitation invokes regular public/private key pair refreshes.

Looking at the results obtained from the tests, it can be observed that the OLSR variations utilising the standard HORS scheme performed the worst out of all the secure OLSR variations tested, in all the different areas tested. This outcome is not surprising, our secure OLSR variations using the standard HORS scheme simply relied too heavily on regular key pair refreshes, which in turn produced too much traffic overhead through the public key distribution process. The requirement of large public key distributions, over varying hop-counts, means the standard HORS OLSR variations are very sensitive to packet loss.

In general, the process of distributing HORS public keys as payloads in OLSR messages performed quite poorly. The large frame sizes associated with a public key distribution, coupled with the need to distribute such public keys on a regular basis, lead to a poor delivery percentage of OLSR messages, most likely due to collisions on the wireless medium. Route establishment times for all hop-counts were high, and route availability was generally very poor. One can conclude that the use of the standard HORS signature scheme, or at least our incorporation of it, as an end-to-end broadcast authentication mechanism for OLSR, is simply not feasible.

### *Extended HORS Signature Scheme*

In an attempt to mitigate the heavy dependency the standard HORS scheme has on key pair refreshes, an extended version of the HORS signature scheme, explained in section 3.5.1, was used as a main signature scheme by four of the secure OLSR variations tested. Without going into too much detail, the extended version of HORS allows a given public key to remain valid for longer, by generating multiple corresponding private keys in a key hash-chain structure. The motivation behind an extended public key lifetime is a reduction in public key distributions, and hence, a reduction in traffic overhead.

In general, the extended HORS scheme performed better than that of the standard HORS scheme. Of the four OLSR variations using the extended HORS scheme, it was found that the extended HORS variations using the longer key chains performed best. As we have discussed numerous times, the process of distributing a HORS public key using OLSR messages is a difficult and lengthly process, particularly over several hop-counts in the presence of packet loss. Whilst the OLSR variations using the extended HORS scheme are still required to distribute HORS public keys, these keys remain valid for longer, and hence can be used to verify more OLSR routing messages signed with a corresponding HORS private key. Of the different parameter sets (HORS 256 and HORS 1024), the HORS 256 parameter set performed significantly better, this is due to the much smaller public key size of the HORS 256 parameter set, making the process of a HORS public key distribution more suitable.

Several drawbacks, however, do exist with the secure OLSR variations using the extended HORS scheme, when compared to the standard HORS scheme: Signature sizes are larger, in fact, twice as large; the storage size of the public/private keys is significantly larger, as a key hash-chain can maintain several private keys; and, the efficiency of signature verifications, using a given public key from a key chain, drops off as more private keys from the key chain are used. Even though the extended HORS scheme performed significantly better than that of the standard scheme, the OLSR variations using the extended HORS schemes still performed worse than the IBS-OLSR variation, and quite substantially worse than the standard OLSR protocol.

## 8.5 RECOMMENDATIONS FOR FUTURE WORK

Throughout the work done thus far, several problem areas have arisen for which improvements could be made. In this section, we aim to to identify some of these problem areas as potential research avenues for improvements. Whilst the areas discussed in the section are not new areas of research - in fact, all are active areas of research - these areas are still relevant to the work that has been undertaken in this dissertation.

### 8.5.1 Reliability in 802.11 Broadcast/Multicast Transmissions

As discussed previously, wireless terminals in 802.11 networks are required to sense the channel prior to transmitting data over it. Unlike wired networks however, the communication boundaries of the wireless technology are not fixed, and are highly variable due to the nature of the medium, making channel sensing ineffective amongst hidden nodes (see appendix A). In 802.11 wireless networks, frame collisions are possible due to such hidden nodes, and frame loss also exists due to the vulnerable nature of the wireless medium.

Owing to such problems facing 802.11 wireless networks, various mechanisms, such as RTS/CTS and frame acknowledgments, have been introduced into the operation of 802.11 in an attempt to mitigate such problems. Such mechanisms, however, are present to enhance the reliability of unicast transmissions and not transmissions of a multicast/broadcast nature. Therefore, a wireless node using a broadcast protocol, such as OLSR, cannot rely on such mechanisms to improve the protocol's communications with other ad hoc neighbours.

Such properties of 802.11 broadcast/multicast transmission makes the process of securing such protocols even more difficult. Securing a protocol more than likely adds overhead, particularly in terms of traffic overhead, which can further expose the unreliability and vulnerability of the underlying 802.11 broadcast system. This is particularly evident if security exchanges take place using broadcast transmissions, as was the case with a HORS public key distribution in the experiments undertaken. Currently, schemes to improve the reliability of the 802.11 multicast/broadcast transmission system are actively being researched.

### 8.5.2 Time Synchronisation in Multi-Hop Ad Hoc Networks

In the absence of assuming time synchronisation amongst nodes in an ad hoc network, the process of time synchronisation is required in order to permit time specific operations amongst nodes, such as timestamp verification in our case. For our experiments, we made use of a very simple loose time synchronisation protocol between a given pair of nodes. As was shown, however, the timestamp verification used in our experiments performed inconsistently and, in general, quite poorly when used over multiple hops.

Time synchronisation in wireless ad hoc networks is a difficult process as there is no central administration to coordinate with. Furthermore, time synchronisation protocols are often subject to security attacks, particularly if intermediate nodes can influence a synchronisation between a given pair of nodes. Considering the environment and circumstances involved in a wireless ad hoc network, the process of creating an end-to-end secure time synchronisation protocol, whilst keeping the protocol as simple as possible, is a very challenging task.

### 8.5.3 Efficient Signature Schemes for Broadcast Authentication

As mentioned in section 2.7.1, conventional digital signature schemes, which are often computationally expensive, may not be practical for use in all environments, particularly ones of limited resources. Efficient asymmetric signature primitives have, for a long while, been an active area of research, with increasing interest in the field due to a rising presence of small electronic devices with embedded systems. The research area of one-time signature schemes is a field which aims to provide efficient and secure asymmetric digital signature schemes.

However, as we have pointed out previously in this work, one-time signature schemes are generally hindered by several limiting factors. The main hindering factor of these schemes, in general, is the limited number of secure signatures a given public/private key pair can provide. Regarding the HORS signature scheme, and other one-time signature schemes of a similar nature, the large key sizes present key management problems, particularly in a decentralised environment such as an ad hoc network. For broadcast routing protocols which may require regular signature generation and verification operations, an efficient authentication mechanism for such broadcast communications is required. Schemes to provide efficient broadcast authentication, whilst providing efficient key management properties, are very much a current area of research.

## 8.6 SUMMARY

Overall, the results of our tests reflected the difficulty involved with adding security mechanisms to a wireless broadcast protocol, such as OLSR, without affecting the efficiency of the protocol severely. The secure OLSR variations that were tested generally performed quite poorly, for various reasons. Often, with experiments done in other works, time synchronisation is assumed between nodes. This assumption however, is not always realistic. This study did not assume time synchronisation, but made use of a loose-time synchronisation protocol amongst communicating nodes, for timestamp verification purposes. However, the timestamp verification procedure resulted in a failure rate that was too high, and was deemed impractical in a multi-hop ad hoc network environment. Our application of the HORS signature scheme, and our extended version of HORS, performed inefficiently as a signature scheme for OLSR, plagued mainly by high traffic overhead and packet loss. It must be stated, however, that the identity based signature scheme performed reasonably well compared to the standard OLSR protocol, especially considering how failed timestamp verifications negatively effected the results. The main drawbacks of the IBS-OLSR variation are the increased traffic overhead it produces, as well as the fact that identity-based signature schemes may not be practical for all ad hoc network environments, as discussed in section 8.4.2 above.

In conclusion, this study has investigated various primitives and potential mechanisms for end-to-end security of the OLSR routing protocol. The process of securing a broadcast ad hoc routing protocol, from end to end, can be highly dependent on the reliability and stability of the intermediate wireless links over which the protocol is applied. The application of such security mechanisms, in the context of broadcast routing protocols for wireless ad hoc networks, will tend to become more practical and viable as improvements are realised regarding the underlying performance of the wireless mediums forming such networks.

APPENDIX A

# HIDDEN NODE PROBLEM

In a Wired network, such as ethernet, stations make use of a multiple access scheme known as Carrier Sense Multiple Access With Collision Detection (CSMA/CD). The CSMA/CD scheme allows stations to determine if another station is transmitting on the medium, as well as detect collisions on the medium.

Unlike a station in a wired network, a wireless station may not, for different reasons, always be able to hear/sense the transmissions of every other node in the wireless network. Hence, due to the nature of the wireless medium, the CSMA/CD scheme cannot be implemented for 802.11 wireless networks. Instead, 802.11 uses a multiple access protocol known as Carrier Sense Multiple Access With Collision Avoidance (CSMA/CA). In CSMA/CA, a station wishing to transmit on the medium must first listen on the channel, for a period of time, to determine if the channel is idle. If the channel is busy, transmission must be deferred. If the channel is sensed as being idle, the station sends a signal informing other stations to not transmit, and then waits a further period of time. If the channel still remains free, the station may transmit.

CSMA/CA however, suffers from a problem known as the hidden node problem [21]. The hidden node problem can arise when wireless stations, not in wireless range of each other, attempt to transmit to a common station.

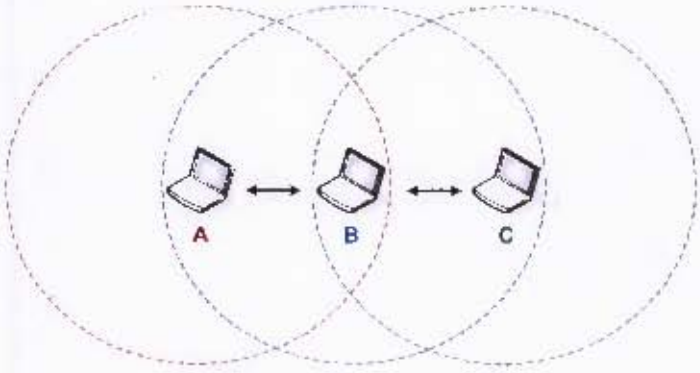



Figure A.1: Hidden Node Problem

Consider the scenario depicted in figure A.1. Stations A and B are within wireless range of each other, as are stations B and C. However, stations A and C lie outside the wireless range of one another, making each of them hidden nodes to the other. Owing to the fact that stations A and C lie out of range of one another, neither can sense the others transmissions on the wireless channel, leaving the CSMA/CA multiple access scheme ineffective in this scenario. This scenario may lead to a situation where stations A and C transmit simultaneously to station B, leading to a potential collision at station B.

To partly rectify this problem, the 802.11 CSMA/CA scheme can be supplemented with a handshake protocol known as Request-to-Send/Clear-to-Send (RTS/CTS) [21]. Whilst RTS/CTS can be used as a potential solution for the hidden node problem, it is a unicast mechanism, and thus cannot be used to protect transmissions of a broadcast nature. For this reason, the RTS/CTS handshake mechanism is not explained here.

APPENDIX B

# SIGNATURE SCHEME VERIFICATION TIMES

Below are the average verification times (in seconds), of the signature schemes tested, over 1000 iterations.

- RSA Signature Scheme: 1.41 seconds
- IBS Signature Scheme: 8.907 seconds

| Private Key in Extended HORS Key Chain | Extended HORS 1024 | Extended HORS 256 |
|---|---|---|
| 1 | 0.227 | 0.267 |
| 2 | 0.347 | 0.413 |
| 3 | 0.460 | 0.563 |
| 4 | 0.577 | 0.713 |
| 5 | 0.697 | 0.860 |
| 6 | 0.810 | 1.003 |
| 7 | 0.930 | 1.153 |
| 8 | 1.053 | 1.300 |
| 9 | 1.163 | 1.450 |
| 10 | 1.287 | 1.600 |

Table B.1: Extended HORS Average Verification Times - Private Keys 1-10

| Private Key in Extended HORS Key Chain | Extended HORS 1024 | Extended HORS 256 |
|---|---|---|
| 11 | 1.403 | 1.750 |
| 12 | 1.520 | 1.890 |
| 13 | 1.640 | 2.043 |
| 14 | 1.753 | 2.190 |
| 15 | 1.873 | 2.333 |
| 16 | 1.993 | 2.483 |
| 17 | 2.113 | 2.633 |
| 18 | 2.230 | 2.777 |
| 19 | 2.347 | 2.923 |
| 20 | 2.467 | 3.073 |
| 21 | 2.583 | 3.223 |
| 22 | 2.700 | 3.370 |
| 23 | 2.823 | 3.513 |
| 24 | 2.937 | 3.663 |
| 25 | 3.060 | 3.813 |
| 26 | 3.177 | 3.960 |
| 27 | 3.293 | 4.107 |
| 28 | 3.410 | 4.250 |
| 29 | 3.530 | 4.407 |
| 30 | 3.650 | 4.550 |
| 31 | 3.770 | 4.700 |
| 32 | 3.887 | 4.850 |
| 33 | 4.003 | 4.993 |
| 34 | 4.120 | 5.137 |
| 35 | 4.240 | 5.293 |
| 36 | 4.353 | 5.440 |
| 37 | 4.473 | 5.587 |
| 38 | 4.590 | 5.730 |
| 39 | 4.710 | 5.880 |
| 40 | 4.830 | 6.027 |

Table B.2: Extended HORS Average Verification Times - Private Keys 11-40

| Private Key in Extended HORS Key Chain | Extended HORS 1024 | Extended HORS 256 |
|---|---|---|
| 41 | 4.950 | 6.177 |
| 42 | 5.067 | 6.323 |
| 43 | 5.187 | 6.470 |
| 44 | 5.303 | 6.620 |
| 45 | 5.420 | 6.770 |
| 46 | 5.540 | 6.913 |
| 47 | 5.653 | 7.063 |
| 48 | 5.770 | 7.210 |
| 49 | 5.887 | 7.360 |
| 50 | 6.010 | 7.500 |
| 51 | 6.127 | 7.657 |
| 52 | 6.243 | 7.803 |
| 53 | 6.363 | 7.947 |
| 54 | 6.483 | 8.093 |
| 55 | 6.600 | 8.243 |
| 56 | 6.717 | 8.390 |
| 57 | 6.837 | 8.547 |
| 58 | 6.950 | 8.687 |
| 59 | 7.070 | 8.837 |
| 60 | 7.187 | 8.987 |
| 61 | 7.303 | 9.133 |
| 62 | 7.427 | 9.280 |
| 63 | 7.547 | 9.430 |
| 64 | 7.660 | 9.577 |
| 65 | 7.783 | 9.723 |
| 66 | 7.897 | 9.870 |
| 67 | 8.020 | 10.017 |
| 68 | 8.137 | 10.167 |
| 69 | 8.257 | 10.307 |
| 70 | 8.370 | 10.463 |

Table B.3: Extended HORS Average Verification Times - Private Keys 41-70

| Private Key in Extended HORS Key Chain | Extended HORS 1024 | Extended HORS 256 |
|---|---|---|
| 71 | 8.490 | 10.610 |
| 72 | 8.607 | 10.757 |
| 73 | 8.727 | 10.907 |
| 74 | 8.843 | 11.053 |
| 75 | 8.963 | 11.200 |
| 76 | 9.077 | 11.350 |
| 77 | 9.197 | 11.497 |
| 78 | 9.317 | 11.643 |
| 79 | 9.430 | 11.790 |
| 80 | 9.547 | 11.937 |
| 81 | 9.670 | 12.093 |
| 82 | 9.793 | 12.240 |
| 83 | 9.910 | 12.390 |
| 84 | 10.027 | 12.533 |
| 85 | 10.143 | 12.673 |
| 86 | 10.260 | 12.830 |
| 87 | 10.377 | 12.977 |
| 88 | 10.497 | 13.117 |
| 89 | 10.613 | 13.273 |
| 90 | 10.733 | 13.420 |

Table B.4: Extended HORS Average Verification Times - Private Keys 71-90

APPENDIX C

# OLSRD MESSAGES AND PACKETS

## C.1 OLSRD DEFAULT STRUCTURES

OLSRd Generic Message

```
struct olsrmsg
{
 olsr_u8_t      olsr_msgtype;
 olsr_u8_t      olsr_vtime;
 olsr_u16_t     olsr_msgsize;
 olsr_u32_t     originator;
 olsr_u8_t      ttl;
 olsr_u8_t      hopcnt;
 olsr_u16_t     seqno;

  union
  {
    struct hellomsg hello;
    struct tcmsg    tc;
    struct hnamsg   hna;
    struct midmsg   mid;
  } message;

}
```

## C.2 OLSRd Custom Structures

Security-Aware OLSR Message

```
struct secure_olsrmsg
{

 olsr_u8_t      signature_algo;
 olsr_u8_t      reserved;
 olsr_u16_t     sig_data_size;

 olsr_u8_t      olsr_msgtype;
 olsr_u8_t      olsr_vtime;
 olsr_u16_t     olsr_msgsize;
 olsr_u32_t     originator;
 olsr_u8_t      ttl;
 olsr_u8_t      hopcnt;
 olsr_u16_t     seqno;

  union
  {
    struct hellomsg hello;
    struct tcmsg    tc;
    struct hnamsg   hna;
    struct midmsg   mid;
  } message;

}
```

Security-Aware OLSR Packet

```
struct secure_olsr_packet
{

 olsr_u16_t     olsr_packlen;
 olsr_u16_t     olsr_seqno;
 olsr_u16_t     encryption_algo;
 olsr_u16_t     reserved;

 struct secure_olsrmsg  secure_olsr_msg[1];

}
```

Loose Time Synchronisation Request Message

```
struct time_request
{
        /*Address of node being requested*/
        olsr_u32_t    requested_host;
        /*Nonce value in request message*/
        olsr_u32_t    nonce;
};

struct time_req_msg
{
        olsr_u8_t     olsr_msgtype;
        olsr_u8_t     olsr_vtime;
        olsr_u16_t    olsr_msgsize;
        olsr_u32_t    originator;
        olsr_u8_t     ttl;
        olsr_u8_t     hopcnt;
        olsr_u16_t    seqno;

        struct time_request trequest;
};
```

Loose Time Synchronisation Response Message

```
struct tstamp_response
{

        olsr_u32_t              request_originator;
        olsr_u32_t              nonce;
        struct timestamp        time_stamp;

};

struct tstamp_resp_msg
{
        olsr_u8_t     olsr_msgtype;
        olsr_u8_t     olsr_vtime;
        olsr_u16_t    olsr_msgsize;
        olsr_u32_t    originator;
        olsr_u8_t     ttl;
        olsr_u8_t     hopcnt;
        olsr_u16_t    seqno;

        struct tstamp_response time_response;
};
```

HORS Public Key Request Message

```
struct hors_request
{

        olsr_u32_t      requested_host;
        olsr_u32_t      nonce;
};

struct hors_request_msg
{
        olsr_u8_t       olsr_msgtype;
        olsr_u8_t       olsr_vtime;
        olsr_u16_t      olsr_msgsize;
        olsr_u32_t      originator;
        olsr_u8_t       ttl;
        olsr_u8_t       hopcnt;
        olsr_u16_t      seqno;

        struct hors_request request_msg;
};
```

HORS Public Key Distribution Message

```
struct hors_pubkey
{

        olsr_u16_t      key_sequence;
        olsr_u8_t       fragment;
        olsr_u8_t       total_fragments;
        char            hors_publickey[1];
};

struct hors_distribution
{
        olsr_u8_t       olsr_msgtype;
        olsr_u8_t       olsr_vtime;
        olsr_u16_t      olsr_msgsize;
        olsr_u32_t      originator;
        olsr_u8_t       ttl;
        olsr_u8_t       hopcnt;
        olsr_u16_t      seqno;

        struct hors_pubkey pub_key;
};
```

# BIBLIOGRAPHY


[1] ADJIH, C., BADEL, M., LAOUITI, A., MHLETAHLER, P., AND PLAKOO, A. Object-orientated OLSR (OOLSR) by team Hipercom from INRIA. Available at http://hipercom.inria.fr/OOLSR/index.html.

[2] AKYILDIZ, I. F., WANG, X., AND WANG, W. Wireless mesh networks: a survey. *Computer Networks 47*, 4 (2005), 445–487.

[3] AL-SHURMAN, M., YOO, S.-M., AND PARK, S. Black hole attack in mobile ad hoc networks. In *ACM Southeast Regional Conference* (2004), S.-M. Yoo and L. H. Etzkorn, Eds., ACM, pp. 96–97.

[4] ALLIANCE, W.-F. Wi-fi certified 802.11n draft 2.0: Longer-range, faster-throughput, multimedia-grade wi-fi networks. available at http://www.wi-fi.org/files/kc/wfa_802_11n_industry_june07.pdf, 2007.

[5] ARGYROUDIS, P., AND O'MAHONY, D. Secure routing for mobile ad hoc networking. *IEEE Communications Surveys and Tutorials 7*, 3 (2005), 2–21.

[6] BADIS, H., CLAVEIROLE, T., AGHA, K. A., FOURATI, A., AND GAWEDZKI, I. Qolyester OLSR Implementation. Available at http://qolsr.lri.fr/code/.

[7] BAEK, J., NEWMARCH, J., SAFAVI-NAINI, R., AND SUSILO, W. A Survey of Identity-Based Cryptography. AUUG'2004.

[8] BLUNK, L., AND VOLLBRECHT, J. PPP Extensible Authentication Protocol (EAP). Internet Request for Comment RFC 2283, Internet Engineering Task Force, Mar. 1998.

[9] BUTTYAN, L., AND HUBAUX, J.-P. Report on a working session on security in wireless ad hoc networks. *Mobile Computing and Communications Review 6*, 4 (Sept. 2002).

[10] CHLAMTAC, I., CONTI, M., AND LIU, J. J.-N. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks 1*, 1 (2003), 13–64.

[11] CLAUSEN, T., AND JACQUET, P. Optimized Link State Routing Protocol (OLSR), Internet Request for Comment (RFC) 3626, October 2003.

[12] CLAUSEN, T., LAOUITI, A., MUHLETHALER, P., RAFFO, D., AND ADJIH, C. Securing the OLSR routing protocol with or without compromised nodes in the network. http://hal.inria.fr/inria-00070513/en/, Feb. 26 2007.

[13] COFFEY, T., AND SAIDHA, P. Non-repudiation with mandatory proof of receipt. *SIGCOMM Comput. Commun. Rev. 26*, 1 (1996), 6–17.

[14] COMMUNICATIONS RESEARCH CENTRE (CRC) CANADA. OLSR Implementation. Available at http://www.crc.ca/.

[15] ERGEN, M. IEEE 802.11 Tutorial. University of California Berkeley. Available at http://www.eecs.berkeley.edu/ ergen/docs/ieee.pdf, July 30 2002.

[16] FEENEY, L. M. A taxonomy for routing protocols in mobile ad hoc networks. Technical Report T99-07, Swedish Institute of Computer Science, Nov. 1, 1999.

[17] FEENEY, L. M., AHLGREN, B., AND WESTERLUND, A. Spontaneous networking: An application-oriented approach to ad hoc networking. Tech. rep., Mar. 30 2001.

[18] FIPS. *Advanced Encryption Standard (AES).* National Institute for Standards and Technology, pub-NIST:adr, Nov. 2001.

[19] FREE SOFTWARE FOUNDATION (FSF). GCC, The GNU Compiler Collection. Official home page located at http://gcc.gnu.org/.

[20] FREE SOFTWARE FOUNDATION (FSF). GNU C Library. Available at http://www.gnu.org/software/libc/manual/.

[21] GAST, M. S. *802.11 Wireless Networks: The Definitive Guide.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[22] GENTOO LINUX™. The Gentoo Linux Operating System. Official home page located at at http://www.gentoo.org/.

[23] HAAS, Z. J. A new routing protocol for the reconfigurable wireless networks. In *In Proceedings of the IEEE International Conference on Universal Personal Communications (ICUPC* (1997), pp. 562–566.

[24] HAAS, Z. J., PEARLMAN, M. R., AND SAMAR, P. Internet Draft: The Bordercast Resolution Protocol (BRP) for Ad Hoc Networks. Available at http://tools.ietf.org/id/draft-ietf-manet-zone-iarp-02.txt, July 2002.

[25] HAAS, Z. J., PEARLMAN, M. R., AND SAMAR, P. Internet Draft: The Interzone Routing Protocol (IERP) for Ad Hoc Networks. Available at http://tools.ietf.org/html/draft-ietf-manet-zone-ierp-02, July 2002.

[26] HAAS, Z. J., PEARLMAN, M. R., AND SAMAR, P. Internet Draft: The Intrazone Routing Protocol (IARP) for Ad Hoc Networks. Available at http://tools.ietf.org/id/draft-ietf-manet-zone-iarp-02.txt, July 2002.

[27] HAFSLUND, A., TNNESEN, A., ROTVIK, R. B., ANDERSSON, J., AND IVIND KURE. Secure extension to the olsr protocol. *2004 OLSR Interop and Workshop, San Diego, CA, USA, August 6-7* (2004).

[28] HE, C., AND MITCHELL, J. C. Security analysis and improvements for IEEE 802.11i. In *NDSS* (2005), The Internet Society.

[29] HOEPER, K., AND GONG, G. Models of Authentication in Ad Hoc Networks and Their Related Network Properties, Technical Report, University of Waterloo, CACR 2004-03.

[30] HONG, F., HONG, L., AND FU, C. Secure olsr. In *AINA '05: Proceedings of the 19th International Conference on Advanced Information Networking and Applications* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 713–718.

[31] HU, Y. C., PERRIG, A., AND JOHNSON, D. B. Packet leashes: a defense against wormhole attacks in wireless networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE* (2003), vol. 3, pp. 1976–1986.

[32] IEEE COMPUTER SOCIETY, 802.11 TASK GROUP. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.*

[33] IEEE COMPUTER SOCIETY, 802.11 WORKING GROUP. IEEE 802.11, The Working Group Setting the Standards for Wireless Local Area Networks (WLANs), http://www.ieee802.org/11/.

[34] IEEE COMPUTER SOCIETY, 802.11i TASK GROUP. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 6: Medium Access Control (MAC) Security Enhancements.*

[35] IEEE COMPUTER SOCIETY, 802.15 WORKING GROUP. IEEE 802.15 Working Group for Wireless Personal Area Networks (WPANs), http://ieee802.org/15/index.html.

[36] IEEE COMPUTER SOCIETY, 802.16 WORKING GROUP. The IEEE 802.16 Working Group on Broadband Wireless Access Standards, http://www.ieee802.org/16/.

[37] JACQUET, P., MINET, P., MÜHLETHALER, P., AND RIVIERRE, N. Increasing Reliability in Cable-Free Radio LANs Low Level Forwarding in HIPERLAN. *Wirel. Pers. Commun. 4*, 1 (1997), 51–63.

[38] JACQUET, P., MÜHLETHALER, P., CLAUSEN, T., LAOUITI, A., QAYYUM, A., AND VIENNOT, L. Optimized link state routing protocol for ad hoc networks. In *Proceedings of the 5th IEEE Multi Topic Conference (INMIC 2001)* (2001).

[39] JOHNSON, D. B., AND MALTZ, D. A. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, Imielinski and Korth, Eds., vol. 353 of *THE KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE*. Kluwer Academic Publishers, 1996.
**http://athos.rutgers.edu/ imielins/book.html**.

[40] JORDAN, R., AND ABDALLAH, C. T. Wireless communications and networking: An overview. available at http://www.eece.unm.edu/controls/papers/jor_cta.ps, 1997.

[41] KENT, S., AND ATKINSON, R. Security Architecture for the Internet Protocol, IETF RFC 2401. Available at http://www.ietf.org/rfc/rfc4301.txt, 1998.

[42] LAMPORT, L. Constructing digital signatures from a one-way function. Tech. Rep. CSL-98, SRI International, Oct. 1979.

[43] LANG, D. A comprehensive overview about selected ad hoc networking routing protocols. available at http://wwwbib.informatik.tu-muenchen.de/infberichte/2003/tum-i0311.pdf. Tech. rep., Masters thesis, Technische Universit at Munchen, 2003.

[44] LIU, C., AND KAISER, J. Survey of mobile ad hoc network routing protocols. Available at http://vts.uni-ulm.de/query/longview.meta.asp?document_id=5346. Tech. rep., Universität Ulm. Fakultät für Informatik.

[45] MERKLE, R. C. A certified digital signature. In *CRYPTO '89: Proceedings on Advances in cryptology* (New York, NY, USA, 1989), Springer-Verlag New York, Inc., pp. 218–238.

[46] NAOR, D., SHENHAV, A., AND WOOL, A. One-time signatures revisited: Have they become practical. http://eprint.iacr.org/2005/442.pdf.

[47] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *FIPS PUB 180-1: Secure Hash Standard.* National Institute for Standards and Technology, pub-NIST:adr, Apr. 1995. Supersedes FIPS PUB 180 1993 May 11.

[48] PAPADIMITRATOS, P., AND HAAS, Z. J. Secure routing for mobile ad hoc networks. In *in SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002* (2002), pp. 27–31.

[49] PEREZ, E. 802.11i (How we got here and where we are headed). Available at http://www.sans.org/reading_room/whitepapers/wireless/1467.php. Tech. rep., SANS Institute, 2004.

[50] PERKINS, C. E. Ad-hoc on-demand distance vector routing. In *In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications* (1999), pp. 90–100.

[51] PERRIG, A. The biba one-time signature and broadcast authentication protocol. In *ACM Conference on Computer and Communications Security* (2001), pp. 28–37.

[52] PERRIG, A., CANETTI, R., TYGAR, J. D., AND SONG, D. The tesla broadcast authentication protocol. *RSA CryptoBytes 5* (2002), 2002.

[53] REYZIN, AND REYZIN. Better than biba: Short one-time signatures with fast signing and verifying. In *ACISP: Information Security and Privacy: Australasian Conference* (2002).

[54] RIGNEY, C., RUBENS, A. C., SIMPSON, W. A., AND WILLENS, S. Remote authentication dial in user service (RADIUS). Internet proposed standard RFC 2865, June 2000.

[55] RIVEST, SHAMIR, AND ADLEMAN. A method for obtaining digital signatures and public-key cryptosystems. *CACM: Communications of the ACM 21* (1978).

[56] SHAMIR. Identity-based cryptosystems and signature schemes. In *CRYPTO: Proceedings of Crypto* (1984).

[57] SHAMUS SOFTWARE LIMITED. The Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL), Shamus Software Limited. Official home page located at http://www.shamus.ie/.

[58] STALLINGS, W. *Network Security Essentials - Applications and Standards*, second ed. Prentice Hall, 2003, pp. 72–79.

[59] THE BLUETOOTH SPECIAL INTEREST GROUP (SIG). The Bluetooth Special Interest Group (SIG), Inc. http://www.bluetooth.com.

[60] THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). The Institute of Electrical and Electronics Engineers, Inc. (IEEE). Http://www.ieee.org/.

[61] TONNESEN, A. Implementing and extending the Optimized Link State Routing Protocol. Available at http://www.olsr.org/docs/master-pres.pdf. Tech. rep., Masters Thesis - Unik University, 2004.

[62] TONNESEN, A., LOPATIC, T., GREDLER, H., PETROVITSCH, B., KAPLAN, A., AND TCKE, S.-O. The OLSR Routing Daemon (OLSRd). Available at http://www.olsr.org/.

[63] TORGERSON, M. D., AND LEEUWEN, B. P. V. Routing data authentication in wireless networks. http://www.osti.gov/servlets/purl/787792-OqS78H/native/, Apr. 16 2007.

[64] VALGRIND. The Valgrind Suite of Tools. Official home page located at http://valgrind.org/.

[65] XU, S., MU, Y., AND SUSILO, W. Secure AODV routing protocol using one-time signature. In *MSN* (2005), X. Jia, J. Wu, and Y. He, Eds., vol. 3794 of *Lecture Notes in Computer Science*, Springer, pp. 288–297.

[66] YANG, H., LUO, H., YE, F., LU, S., AND ZHANG, L. Security in mobile ad hoc networks: Challenges and solutions (2004). IEEE Wireless Communications. 11 (1), pp. 38-47. Postprint available free at: http://repositories.cdlib.org/postprints/618.

[67] ZAPATA, AND ASOKAN. Securing ad hoc routing protocols. In *WiSe: Proceedings of the ACM Workshop on Wireless Security* (2002).

[68] ZAPATA, M. G. Secure ad hoc on-demand distance vector routing. *Mobile Computing and Communications Review 6*, 3 (2002), 106–107.

[69] ZHAN, M. Reliable Multicast Extension to IEEE 802.11 in Ad Hoc Networks. Available at ftp://fas.sfu.ca/pub/cs/TH/2003/MeihuaZhanMSc.ps.gz. Tech. rep., Masters Thesis - Simon Fraser University, 2003.

[70] ZHANG, K. Efficient protocols for signing routing messages. In *NDSS* (1998), The Internet Society.

[71] ZHOU, L., AND HAAS, Z. J. Securing ad hoc networks. *IEEE Network 13*, 6 (1999), 24–30.

[72] ZOU, X., RAMAMURTHY, B., AND MAGLIVERAS, S. Routing techniques in wireless ad hoc networks-classification and comparison. In *6th World Multiconference on Systemics, Cybernetics and Informatics. Proceedings*. Int. Inst. Inf. & Syst, 2002.