Efficient Security Management for Active Networks

Lawrence L. L. Cheng

Submitted to the Department of Electrical Engineering in fulfilment of the requirements for the degree of Doctor of Philosophy in Electrical Engineering

University College London (UCL)

2007

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

DECLARATION

This is to declare the work presented in this thesis is the author's own work.

ACKNOWLEDGEMENT

The author of this thesis would like to thank Prof. C. Todd, Prof. A. Galis for their valuable advice, and generous help for supporting the research work presented in this thesis.

To Stanley and Sandra, the giants on whose shoulders I stand.

ABSTRACT

Due to the dynamic nature and dynamic routing capability of active packets, security in active networks should be hop-by-hop based. This thesis discusses the identified drawbacks of existing approaches. These drawbacks are: the high performance overhead generated by per-hop Security Association (SA) negotiation prior to secured active packet transmission; the high complexity in SA negotiation handshake process; active packet can only be securely transmitted after SA negotiations; the shared key set generated for protecting active packets may not have Perfect Forward Secrecy (PFS); lack of confidentiality protection on exchanged symmetric keys and active packets; lack of SA negotiation power; and scalability issues. This thesis presents a novel hop-by-hop active network security management approach known as Security Protocol for Active Networks (SPAN). SPAN is designed to enable secure active packet transmission *during* a series of hop-by-hop SPAN SA negotiation along a new execution path, instead of after. The design of SPAN has taken into consideration the factors of security, efficiency, flexibility, scalability, and applicability. SPAN is resistant to replay, man-in-the-middle, impersonate attacks. SPAN is designed to detect DoS attacks much more efficiently. Furthermore, SPAN is uniquely designed to enhance the robustness and efficiency of underlying active networking systems.

INDEX

ABBREVATIONS

AA	Active Applications
ACK	Acknowledgement
AH	Authentication Header
AN	Active Networks
ANEP	Active Network Encapsulation Protocol
ANTS	Active Node Transfer System
API	Application Programming Interface
ASV	Authentication Server
CA	Certificate Authority
CBC	Cipher Block Chaining
CPU	Central Processing Unit
CRL	Certificate Revocation List
D-H	Diffie-Hellman
DARPA	Defence Advanced Research Projects Agency
DES	Data Encryption Standard
DoS	Denial of Service
DS	Digital Signature
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
EE	Execution Environment
ESP	Encapsulating Security Header
EU	European Union
FAIN	Future Active IP Networks
HMAC	Hashed Message Authentication Code
I	Initiator

I-D	Internet Draft
ICMP	Internet Control Message Protocol
IKE	Internet Key Exchange
IP	Internet Protocol
IPSec	IP Security
ISP	Internet Service Provider
IST	Information Society Technologies
IV	Initialisation Vector
JFK	Just-Fast-Keying
KMM	Key Management Module
KSV	Keying Server
MAC	Message Authentication Code
MD	Message Digest
MD5	Message Digest #5
MSEC	Multicast Security
MTU	Maximum Transfer Unit
NodeOS	Node Operating System
OS	Operating System
OSI	Open Systems Interconnect (model)
P2P	Peer-to-Peer
PFS	Perfect Forward Secrecy
PLAN	A Packet Language for Active Networks
PMTU	Path Maximum Transfer Unit
PKI	Public Key Infrastructure
PRF	Pseudo-Random Function

R	Responder
RFC	Request For Comments
RSA	Rivert Shamir Adelman
SA	Security Association
SADB	Security Association Database
SANE	Secure Active Network Environment
SANTS	Secure Active Node Transfer System
SHA	Secure Hash Algorithm
SKIP	Simple Key Management for Internet Protocol
SKT	Signed Key Transport
SNAP	Safe and Nimble Active Packet
SNMP	Simple Network Management Protocol
SPAN	Security Protocol for Active Networks
SPD	Security Policy Database
SPI	Security Parameter Index
SYN	Synchronisation
TCP	Transmission Control Protocol
TDES	Tripe DES
TS	Traffic Selector
TTL	Time-To-Live
UDP	User Datagram Protocol
XOR	Exclusive-OR

1 Introduction & Background

1.1 An Overview

Today's Internet consists of millions of interconnected nodes, which are divided in interconnected domains that are managed by different Internet Service Providers (ISPs). Connections are typically made between two (or more) end points that are located at the edges of the network. High-speed routers are used in the core network simply as packet routing devices that route packets to their destinations. Figure 1 shows a typical example connection that allows a user to access a file server in his office in today's Internet.

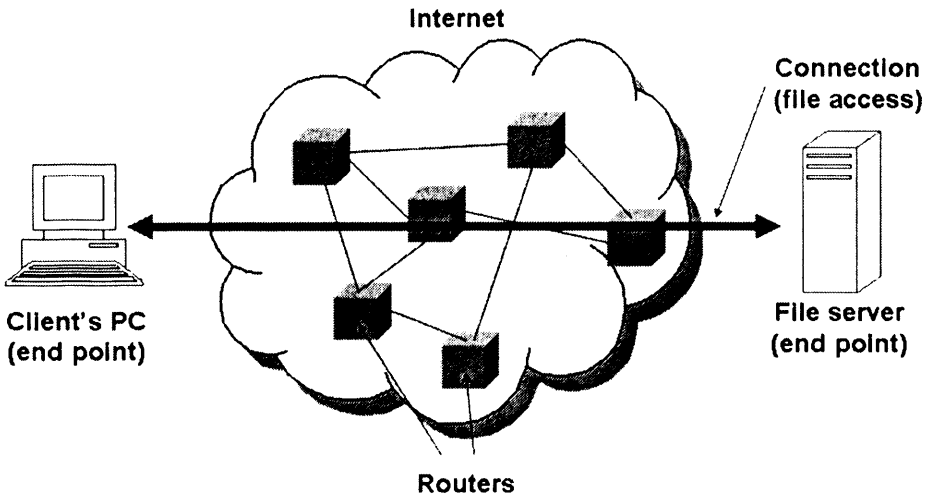

Figure 1 – A typical connection in the Internet

Over the years, the possibilities of utilising the available resources on routers (that are currently being used as packet forwarding machines) have been under

investigation. Active networking, or the concept of Active Networks (ANs), was first described in [1]. In [1] and [2], the authors discussed respectively that active networking technologies would allow active network users to launch their customised executable code (in the form of active packets) to nodes in the network. In addition, the benefit of introducing active networking technologies to the Internet was highlighted: a range of new applications (e.g. new Internet services) would be enabled through the decoupling of services from the underlying architecture. It was further discussed in [3] that active technologies could be interpreted as a means to enable the network to carry out the role of a computer, in order to support a wider range of (new) services in the network. Figure 2 shows the concept of (new) service deployment via active technologies, in which a network administrator may configure or customise, via active management applications, the features of the networks via active technologies, in order to create a tailor environment to support a range of services (e.g. a QoS-guaranteed access to a remote file server). In other words, active technologies "open up" the resources that are currently available in the Internet to network users; hence provide the opportunities to create and support customised (new) services in the Internet.

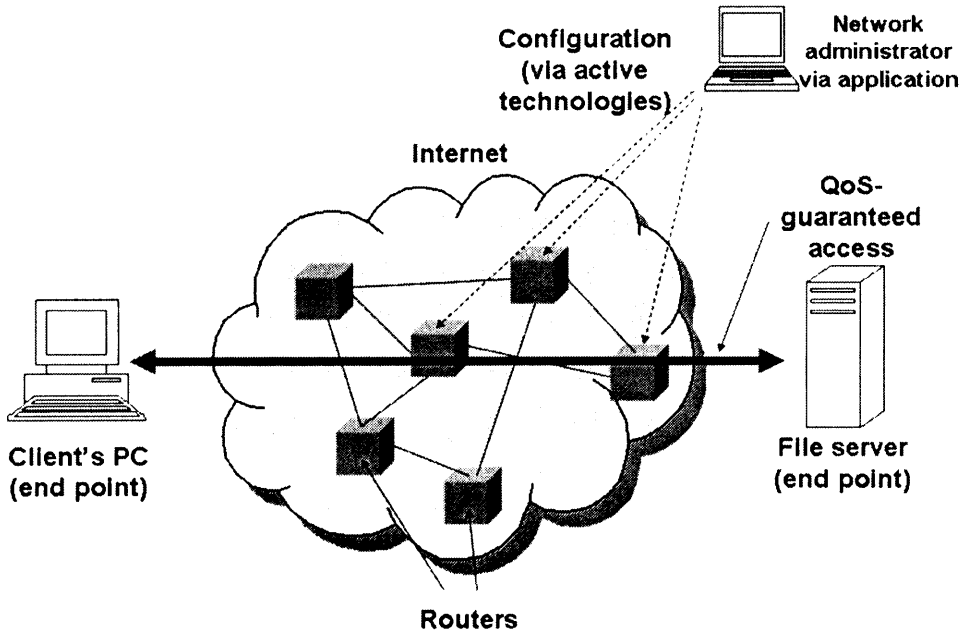

Figure 2 – (New) service deployment via active technologies

More specifically, active technologies could potentially allow a party to install its own programs and run services on any nodes in the network, in a way similar to how programs are installed and run on computers today. Installed programs are then executed in order to support the operations of (new) network-wide services.

Since the introduction of the concept of active networking, a series of active networking programs/projects were funded by the Defence Advanced Research Projects Agency (DARPA) [4]. Several working groups were created within DARPA active networks program to address various architectural issues in active networking, such as the AN Node Operating System (NodeOS) Working Group [5], the AN Security Working Group [6], and the AN Composable Services Working Group [7]. A document [8] that describes the consolidated

work of the DARPA active networks program working groups was prepared as a draft Request For Comments (RFCs). A survey that describes the fundamental concepts of active networking, and the research work carried out during the introduction stage of the active networking concept, can be found in [9].

There are currently two implementation approaches to realise the concept of active networks: the in-band approach and the out-of-band approach. The in-band approach involves the use of discrete capsules which are essentially executable programs (or code) encapsulated in traditional data carrying packets (such as IP packets). These packets - known as active packets - are intercepted and executed at active nodes along the path [10]. In contrast, the out-of-band approach involves the use of existing packet format, and the use of (external) separated mechanisms, to install additional functionalities on routers in a dynamic fashion. For example, the active extension switchlets [11] are sent to nodes in the network, which then load new services onto the nodes to enable the processing of other switchlets. Note that the fundamental concept of in-band and out-of-band approaches is the same: existing routers' functionalities are extended to process packets carrying some form of control code.

In order to use any new technology, the related security threats must be identified and addressed. Thus, the discussion of the use of active technologies for supporting (new) services in the network would be incomplete without an investigation of the relevant security issues. A known security issue of active networks, i.e. hop-by-hop protection for active packets, is addressed in this thesis. Hop-by-hop security is in contrast to traditional end-to-end security in today's Internet. Normally, packets are sent in an end-to-end fashion (Figure 1).

A packet is sent by a client, and received by a server (vice versa). In this case, only two nodes located at either end of a communication link are involved. The intermediate nodes (i.e. the nodes in-between the client and the server) simply route the packet to its destination. The need for hop-by-hop security in active networks is due to a unique feature of active networks: in contrast to *passive networks*[1] that route data packets between end nodes (section 8.11 on p.179), active networks use control packets that may change state as they traverse the network. Intermediate nodes (i.e. the routers in Figure 1) are no longer simple packet forwarding machines, but they are now packet intercepting and forwarding machines, that intercept active packets on the wire, execute the code carried in the packets, and (optionally) add the execution results to the packets before forwarding the packets to their destination. The state-changing feature of active packets traversing the insecure Internet creates new security challenges.

1.2 Motivations

This thesis reports on an investigation to develop a secure, efficient, flexible, and scalable hop-by-hop security solution for protecting the authenticity, integrity, confidentiality, and non-repudiation [12] of active packets that are transmitted in a hop-by-hop fashion in active networks.

The author of this thesis was an active member of the European Union-Information Society Technologies (EU-IST) active networking research project, i.e. Future Active IP Networks (FAIN) [13], during 2001-2004. The author was a member of the security architecture group and the active node architecture prototype development group. The author's incentive to develop

[1] A passive network is one that transmits data packets through passive nodes (e.g. routers) only.

hop-by-hop security systems was initiated during his participation in the FAIN project.

1.3 Objectives

The reason for developing a hop-by-hop security system for active networks was to protect active packets. Note that in contrast to conventional *passive packets* (section 8.11 on p.179), active packets are dynamic (i.e. that they may change state at each intercepting active node during transmission). Thus, instead of using end-to-end security techniques that protect passive packets (that do not change state during transmission), hop-by-hop security is required. Due to the scope and nature of this thesis and space limitation, this thesis shall neither discuss the motivation of active technologies, nor justify the impact of active technologies. Note that this thesis investigates security management; as such, discussions on technological advances in cryptographic algorithms are out-of-scope in this thesis. However, relevant security terms are presented in the Appendix, and all security terms are referenced.

1.4 Thesis Structure and Organisation

This thesis is organised as follow: first, the fundamental concepts of active networks will be presented. Then, one of the major security challenges of active networks, namely, hop-by-hop security, is addressed. This is followed by a detail discussion of the advantages and limitations of existing solutions for hop-by-hop security in active networks. Then, the author's solution, SPAN, will be presented. The SPAN protocol will be discussed and evaluated against relevant existing solutions. This thesis ends with a conclusion, future work, and appendix.

1.5 Active Networks

1.5.1 An Overview

An active network does not exist on its own, it is meant to co-exist with existing networks such as the Internet i.e. a passive network. An active network, consists of both passive nodes and active nodes. An active node is a passive node with an active platform installed. An active node transmits and processes active packets that carry executable active code. Active codes in active packets are executed on intermediate active nodes to perform various tasks. Figure 3 shows an example active network.

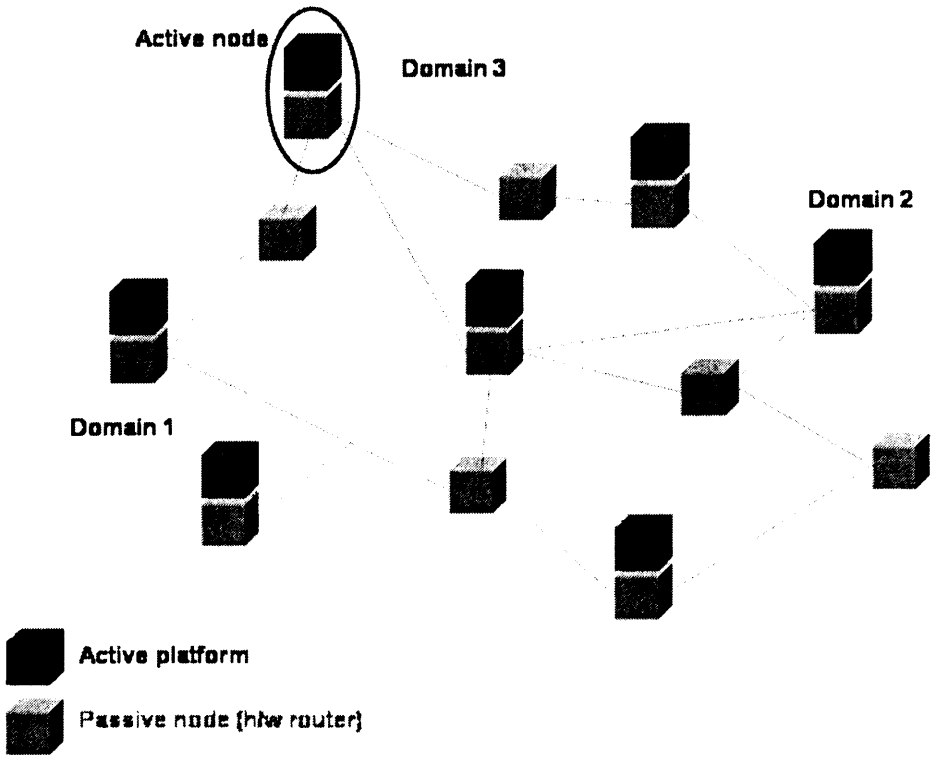

Figure 3 – An example active network

The point to note is that an active network is a *store-compute-and-forward* network (in contrast to a store-and-forward passive network). An active network is composed of a mixture of interconnected active and passive nodes. Active nodes across heterogeneous administrative network domains inter-work with

each other. Active packets (which carry executable active code) are injected into the active network, and are intercepted at desired intermediate active nodes. Active codes in active packets are executed at intermediate active nodes, *before* the packets are forwarded to their next hop. The execution of active code on active nodes creates the opportunity for new service deployment in today's networks. In section 1.5.3 (p.24), an example will be given to show how active networks may operate in practise.

1.5.2 Common Terms in Active Networks

Some terms that are commonly used in active networks are defined below:

- Active Packets

These are special type of IP packets. Active packets use standard transmission protocol such as User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). Active Network Encapsulation Protocol[2] (ANEP) [14][15] is defined as the active packet header protocol. ANEP packets are carried within UDP packets. Active packets' content, i.e. active static code and dynamic data (section 1.6 on p.28), are determined and executed at active nodes.

- Active Code

There are executable codes that are carried in active packets. They can be any type of executable code ranging from programs written in assembly code [16], or programs implemented as Java classes [17], or Simple Network Management Protocol (SNMP) commands.

- Active Node

An active node is composed of a passive node and an active platform. If a

[2] ANEP is the work of many researchers from different institutions researching active networks.. The purpose of defining ANEP is to specify a mechanism for encapsulating Active Network frames for transmission over existing network infrastructure such as IP and IPv6. At the time of writing, ANEP is defined as a draft RFC.

hardware passive router is used as the underlying router, a computer is attached to the hardware router (e.g. through Ethernet cables). An Operating System (OS) and the active platform (e.g. some special software) would be installed on the computer. The active platform can then control the hardware passive router. If software passive routers are used instead, an active platform will be installed on the computer, and routing will be carried out by the OS installed on the computer.

Active nodes are also capable of forwarding packets just like passive nodes. In addition to packet forwarding, active nodes are capable of intercepting active packets, investigating and executing the active code carried in the packet, and performing various computational tasks as specified in the active code. Optionally, execution results may be added back to the packet before the packet is forwarded to its next hop.

- Active Platform

Over the last few years, a general architecture for active networks has evolved [18]. In general, this architecture, i.e. an active platform, is a software platform that is capable of interpreting active packets on the wire, and executing the active code carried in the active packet. An active platform enables an active node to possess the node resources to compute various computational or operational tasks.

In [19], a generalised architecture for active platform was presented. This architecture identifies three layers of code running (Figure 4). The lowest layer is the NodeOS, which hosts several support services such as resource control, security, and packet (de)multiplexing. On top of these support services are Execution Environments (EEs), which can be considered as resource

abstractions for supporting service execution, deployment, configuration, and more. On top of the EEs are Active Applications (AAs), such as end user service applications and management applications. AAs make use of the services and resources on the node, which are made available through the interaction between the EEs and the NodeOS.

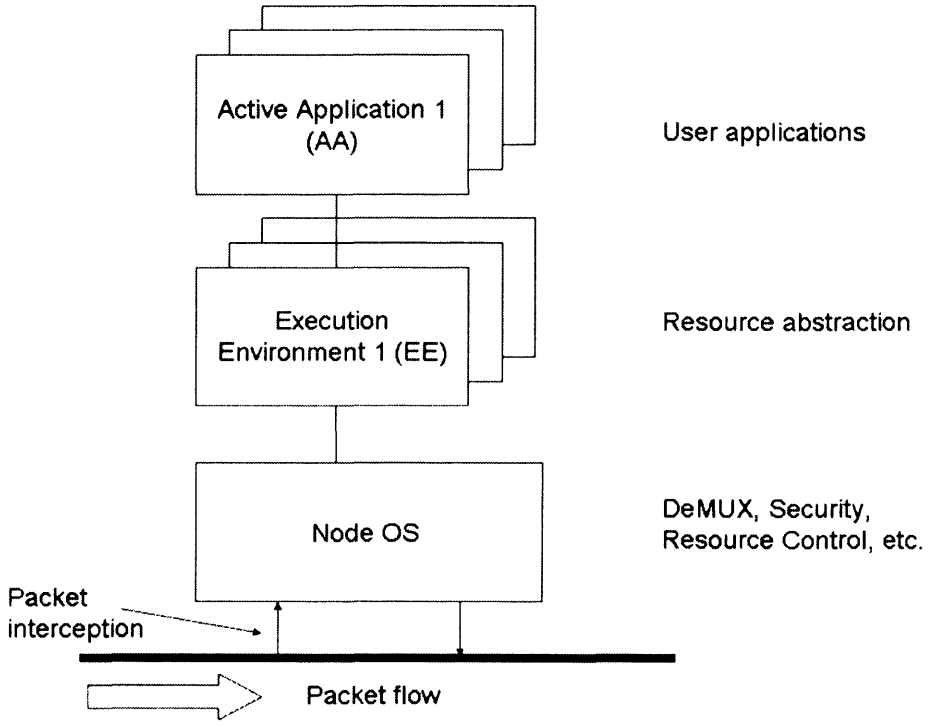

Figure 4 - The generalised active node architecture

- NodeOS

The NodeOS abstracts the hardware (i.e. the router), and provides a range of low-level management facilities to support the operations of the EEs (and subsequently the AAs) [20]. The communications between the EEs and the NodeOS are conducted through a set of Application Programming Interfaces (APIs). Example management facilities are packet multiplexing (i.e. intercepting and examining packets from the wire), security checks on intercept packets (e.g.

authentication and integrity checks), and resource control (e.g. outgoing network bandwidth, memory access control).

- Execution Environments

EEs implement a very board definition of a network API, ranging from programming languages to virtual machine [21]. Example EEs are the Packet Language for Active Networks (PLAN) [22], Active Node Transfer System (ANTS) [23], and the Future Active IP Networks (FAIN) component-based, and more. In general terms, an EE is an active network's programming environment, that when instantiated, it is a runtime environment for the execution of active code (that are carried by active packets, which are intercepted by the NodeOS from the wire). Thus, an EE may be viewed as a definition of a specific programming model for the development of a specific AA. An EE may implement a set of resource abstractions, using the building blocks as provided by the APIs, which link the NodeOS and the EE. To create a service, one may manipulate the set of abstractions that are implemented by the EE.

- Active Applications

An AA is a user-space application, which provides services to users of active networks. An AA may trigger code to be downloaded into active nodes, which subsequently customise the network to support its needs. An example AA could be a QoS-guaranteed media delivery application, which triggers the active platform to reserve certain amount of bandwidth along the path of a media stream (by injecting active code into the networks).

1.5.3 An Example Active Network Operation

A simple example that shows how an active network may operate is shown in Figure 5. The purpose of presenting this example is to give the readers a better

understanding of the key features of active networks[3]. This example is chosen because it highlights certain features of active networking, which will enable the readers to identify the contrasts between the features of active networks and passive networks. For example, active networks have the flexibility to support dynamic code execution; the capability of supporting (new) service deployment in the network; and most importantly, active networks may change the states of active packets.

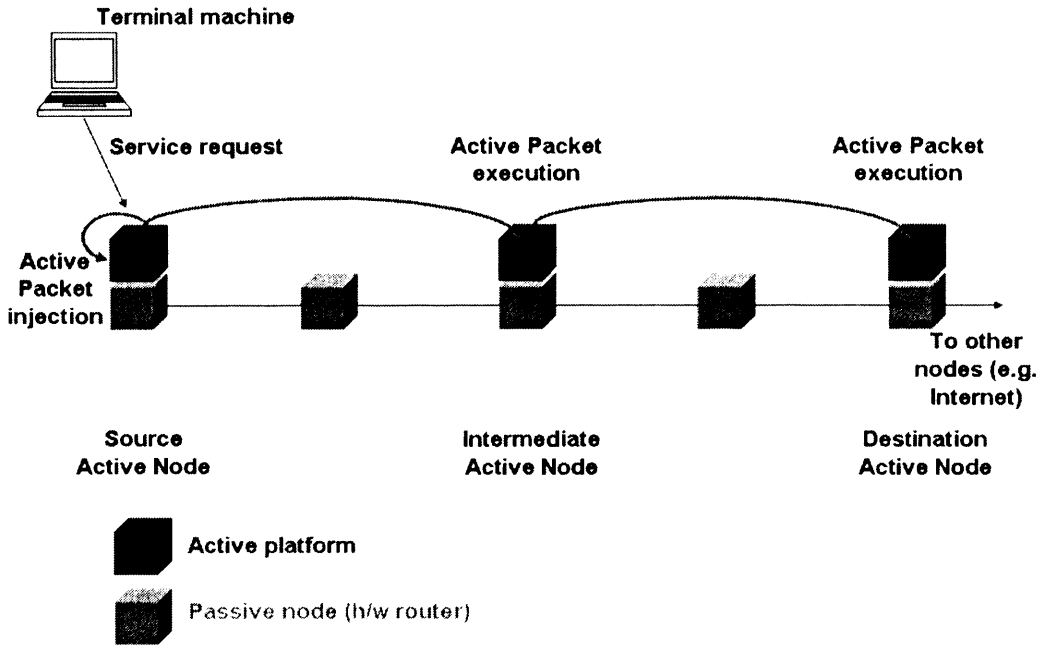

Figure 5 – An example of active network operation

Figure 5 shows an active network that consists of both passive and active nodes, which is connected to the Internet. The scenario is that the network administrator (which generates commands from a terminal machine) would like to get the IP addresses of the traversed active nodes of an active packet via an

[3] This example is based on the demonstration that the author presented at the FAIN project audit in March 2003.

active management application. Traditionally in passive networks, the administrator/application may issue a traceroute command, which determines the packets' route. The fundamental concept of traceroute is to send traceroute packets to a particular destination on the network, which triggers bypassing nodes to send Internet Control Message Protocol (ICMP) messages to the original sender (in this case, the terminal machine). These ICMP messages are used by the traceroute program to generate a list of hosts through which the packets have traversed en route to the destination.

In the active network scenario, the management application, which is located on a terminal machine (operated by, say, the network administrator), triggers the following procedures:

1. An active packet that carries executable active code is injected at the source active node. The executable active code (when executed) is capable of retrieving the IP address of the active node that the active packet is currently residing. The active code, in this example, is a Java class that invokes a particular SNMP GET command that gets the IP address of the node.
2. The packet is then intercepted on each intermediate active node. Note that there are also passive nodes residing along the communication path. The packet is passed onto its next hop if intercepted by a passive node. As far as the passive nodes are concerned, an active packet is the same as an IP packet. This is because active technologies do not override or replace IP; for example, an active packet carrying small size static code is encapsulated into the payload of a UDP packet prior to packet transmission[4].

[4] This arrangement applies to transmitting small size static code only. If the size of the static code was large i.e. too large to fit into the size of an UDP packet, an alternative approach would be to put a network location reference (e.g. IP address) in the active packet, instead of the actual static code. The reference refers to a network location where the static code could be downloaded. This

3. The code carried in the active packet (which gets the IP address of the traversed active node) is executed on each intermediate active node. The result of code execution (the IP address of the traversed active node) is added back to the active packet.
4. The active packet will eventually carry a list of IP addresses. After traversing a certain number of hops[5], the packet is returned to the source node by the active platform (the original source address is obtainable from the code). The network administrator retrieves the information (i.e. a list of IP addresses) through the management application. Figure 6 shows how the content of the active packet changes when the packet traverses the network.

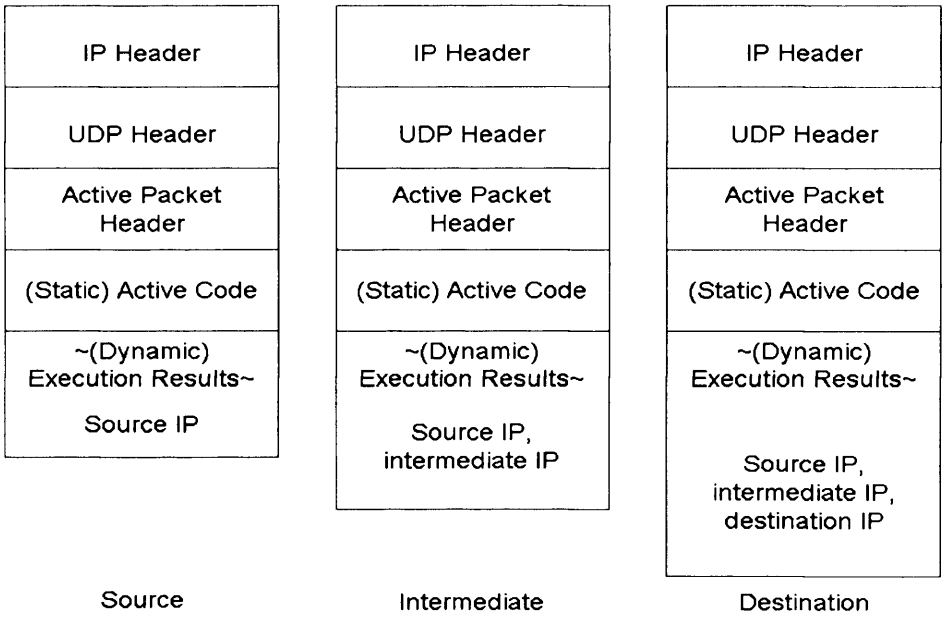

Figure 6 – Changes in packet content

---

is an example of an out-of-band approach in active networks [14].

[5] The Time-to-Live (TTL) value in this demonstration was set to 10 hops, which means the active node will stop forwarding the packet after it has traversed 10 hops. This value is needed to prevent packets from looping indefinitely in the network. The value may be adjusted to include more hops, the current standard is 255 hops for passive packets.

This scenario shows how active networks operate differently from passive networks; but at the same time, they may co-exist with passive networks. Instead of triggering hosts to drop the original packet and to send ICMP messages back to the original sender, active nodes may dynamically execute an active packet, add contents to the active packet, and route the packet back to its original source. These unique capabilities of active networking show that active networks are much more dynamic and flexible than passive networks. This high level of dynamicity and flexibility of active networks creates a new range of security challenges for active network developers.

1.6 Features of Active Networks

1.6.1 Dynamic Data and Static Code in Active Packets

The above example illustrates how active technologies may operate. The key point to note is that the content of an active packet may *change* whilst it is crossing the network. Figure 6 shows that in active networks, the result(s) of code execution is(are) added back to the packet before the packet is forwarded to its next hop. Using the example in section 1.5.3 (p.24), a new IP address is appended to the list of traversed IP addresses that are carried in the active packet, each time the active code is executed on an active node. The state-changing feature of active packets is known as the dynamic nature of active packets in this thesis.

Another point to note is that besides dynamic data, active packets carry *static code*. Static code refers to the executable active code that is generated at the source node prior to packet injection. The code is static in the sense that it is not to be modified when the packet traverses through the network Active packets may carry dynamic code: the term dynamic code means that the code injected

into the active network at the source node may be modified at other nodes. The reasons for generating dynamic code (or more specifically, modifying the original code) could be, for example, when an active node believes the original code is no longer suitable for its originally designed purpose (e.g. new network conditions may cause active network operators to generate new code to accommodate the new conditions); or it may want to add additional commands to the packet to fit its own needs. At the time of writing, the author has not come across the use of dynamic code in active networks. Thus, this thesis does not discuss the use of dynamic code in active networks. However, the security protocol presented in this thesis can be used to protect active packets carrying static code or dynamic code (section 3.9 on p.94).

1.6.2 Dynamic Routing in Active Networks

Note also that active packets support dynamic routing i.e. the route is not known in advance (at the time when the active packet is injected at the source node). This is because the next hop of execution may depend on the execution results [16][17][23], therefore only the node that a packet execution has just taken place would be able to decide where the executed packet should be forwarded to (Figure 7). Dynamic routing has a major impact on the design of a hop-by-hop security solution for active networks.

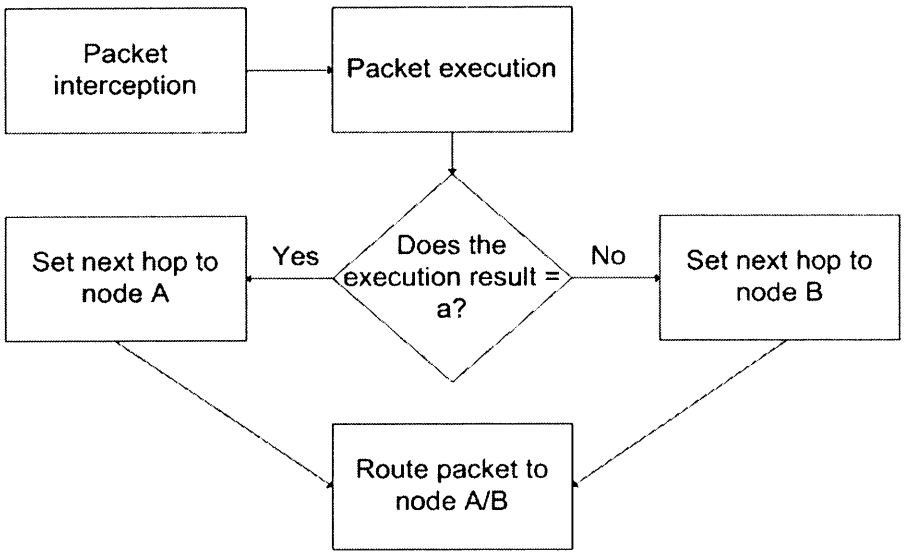

Figure 7 – Dynamic routing at an active node

Question 1: What are the difficulties of defining an exact route for an active packet at the point of packet injection at the source node?

Imagine a traveller would like to specify the fastest travelling route from one place to another on the London Underground, prior to beginning his/her journey. The traveller could specify a route if he/she had a (London Underground) map in hand (i.e. pre-knowledge of the entire network), and he/she had access to real-time congestion information of route(s)/station(s) (i.e. real-time network status). Then, the traveller could determine all possible routes, and determine the fastest route for his/her journey. Similarly, specifying a static execution route for an active packet is only possible when - prior to active packet injection - the administrator/management application is capable of specifying exactly at which nodes the packet should be executed. This requires the administrator/management application to have in-depth (real-time) knowledge of the managed network, and the network status remains static (e.g. no new congestion in the network). This requirement is not scalable because it cannot

be justified when the managed network is large, for example the Internet, which consists of millions of nodes. Furthermore, the network status may change in real-time (i.e. congestion may appear at any point and at any time).

Specifying a static execution route limits the scalability and flexibility of active packet routing. To enhance flexibility, active platforms should be capable of dynamically re-routing active packets, depending on real-time network status.

1.6.3 Hop-by-Hop Transmission

Because active packets may change state at each hop, this thesis refers this transmission model as the *hop-by-hop transmission model* [24][25][26]. Hop-by-hop transmission model is in contrast to the end-to-end transmission model of passive packets, the latter model involves packets that do not change state during their traversal through the network. A hop-by-hop transmission model is applicable to active packets due to the dynamic nature and dynamic routing capabilities of active packets. A hop-by-hop active packet transmission involves exchanging messages between pairs of nodes along the packet's path; the node that sends the packet is the Initiator (section 8.9 on p.178), the packet receiver is the Responder (section 8.15 on p.189). An Initiator is a node where the *principal*[6] resides. An example Initiator would be the source node of an active packet. A Responder would be a node that is about to receive an active packet from the Initiator e.g. the next hop of packet forwarding.

[6] A principal is the actual creator of executable active code. For example, an administrative or management application that creates active packets. Active packets contain static code that is executed for control or management purposes.

1.7 Security in Active Networks

1.7.1 Hop-by-Hop Security

Given that active packets may change state during network traversal, there is a need to protect the authenticity, integrity, confidentially and non-repudiation of active packets, in a hop-by-hop manner [27][28]. Example threats that are applicable to active packets are replay attacks, impersonate attacks, man-in-the-middle attacks, and DoS attacks [29] (see section 4.2 on p.100 for more detail on these attacks, and how the author's solution addresses these attacks). This thesis refers this security approach as a hop-by-hop security model. Figure 8 shows the deployment of a hop-by-hop security model in active networks. Note that two security tunnels are deployed in Figure 8: tunnel a is established between the source active node and the intermediate active node; whereas tunnel b is established between the intermediate active node and the destination active node. These tunnels are established separately in the network to enable a hop-by-hop transmission of active packets. Note further that in contrast, in passive networks, tunnels (e.g. tunnel z) are usually established in an end-to-end fashion.

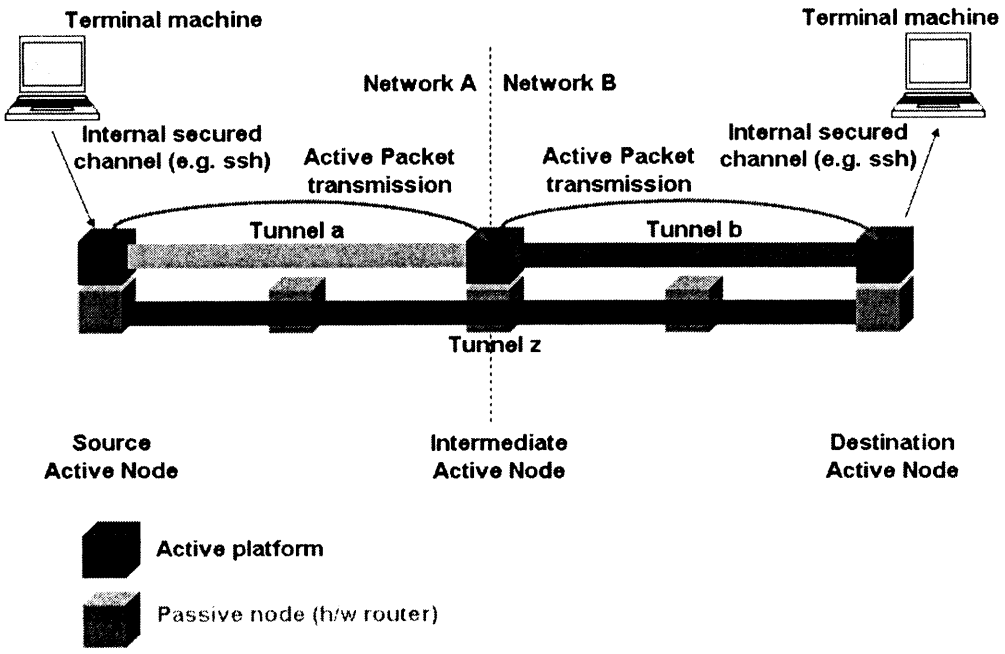

Figure 8 – Hop-by-hop security Vs end-to-end security

Hop-by-hop security is needed because:

- The authenticity and integrity of static code must by verified at each packet intercepting node, based on the principal's authenticity. This is because the static code should be verified based on the identity of its *actual* creator. Without hop-by-hop authenticity and integrity protection, an attacker may change the ownership or contents of the static code in active packets, resulting in compromised code execution on active nodes.
- Static code must also be subjected to non-repudiation protection, so that the principal cannot deny of any wrongdoing should code execution compromises other nodes in the network. Otherwise, anyone may create and inject miscellaneous active packets, and deny any wrongdoing.
- The authenticity and integrity of the dynamic data carried in a packet (i.e.

the execution results, or other information that an intermediate node adds to the executed active packet, and wishes to pass onto its next hop) should be verified based on the identity of the entity that the packet was *last modified* (i.e. based on the identity of the last execution node). This is because the data should be verified based on the identity of its actual creator. Integrity protection is important, otherwise attackers may change the content of active packets, resulting in compromised code execution.

- Furthermore, the entire packet should be subjected to confidentiality protection; otherwise, intruders may obtain potentially sensitive information from the dynamic data on packets, such as code execution results.

Question 2: Is it beneficial to protect *all* (active and passive) packets with hop-by-hop protection?

The *contents* of passive packets would not change whilst the packets are in transit, only the hop count value in their headers would. Protecting the static headers and contents of passive packets is a form of end-to-end protection.

In active networks, the identity of the intermediate nodes between the source and the destination is important. This is because the data carried in an active packet is expected to be modified at some intermediate nodes during packet transmission. Upon receiving an active packet, each intermediate active router must verify:

- The integrity of the packet arriving from its neighbouring active router.
- The authenticity of its neighbouring active router (at which the active packet was executed and modified, and where the packet was delivered from).

- The authenticity and integrity of the source active router.

1.7.2 Challenges in Designing Security Solutions for Active Networks

This thesis suggests that the challenge in hop-by-hop protection is that there is a need to find a balance point between security and performance. For example, it may appear that the simplest solution to hop-by-hop protection is to digitally sign the modified parts of an active packet at each router (that packet modification has taken place). However, this is not practical. This is because asymmetric operations are much slower than symmetric operations (Question 6 on p.166). The scalability and efficiency evaluations on different asymmetric and symmetric approaches for protecting active packets in a hop-by-hop manner are presented in a later chapter of this thesis (section 5.4 on p.140). Using symmetric keys would be an alternative, but because there is no centralised authority that distributes symmetric keys - in contrast to asymmetric keys where several well known Certificate Authorities (CAs) (section 8.1 on p.162) exist - the use of symmetric keys in a hop-by-hop environment requires an efficient key distribution mechanism to generate keys between hops. Also, note that active packets support dynamic routing. This implies that the route is unknown at the point of packet injection, and further implies that a hop-by-hop security system must be capable of *dynamically* and *efficiently* setting up *Security Associations (SAs)* (section 8.16 on p.189) between hops. A SA describes a set of security parameters that are needed for maintaining or operating a security channel.

Furthermore, existing active network systems assume compatibility. Existing applications in active networking systems currently assume that a piece of active code injected to the network can be executable on all other remote nodes.

This is, obviously, an assumption that would not hold in a large-scale network. Thus, this thesis will also investigate a secure solution to ensure that, prior to establishing a SA, and sending across active packets, the communicating peers must ensure compatibility between themselves in a secured fashion.

Note that this thesis is investigating a secure solution for efficient hop-by-hop SA negotiation (and hence subsequent secure packet transmission) along a *new* execution path. This thesis defines a new execution path to be a path which no active packets have previously traversed (hence no pre-established hop-by-hop SAs), or a path of which previously established hop-by-hop SAs has expired, therefore there is a need to establish a series of new hop-by-hop SAs along the path (hence the name new execution path). In contrast, an old execution path would be one where active packets have previously traversed (hence hop-by-hop SAs have been established); and the hop-by-hop SAs have not expired (so there is no need to renew or re-establish hop-by-hop SA).

2 State-of-the-Art in Hop-by-Hop Security

Existing active network security systems for hop-by-hop protection and related work are discussed in this chapter. Their features and drawbacks are identified.

2.1 Symmetric Cryptography for Hop-by-hop Security

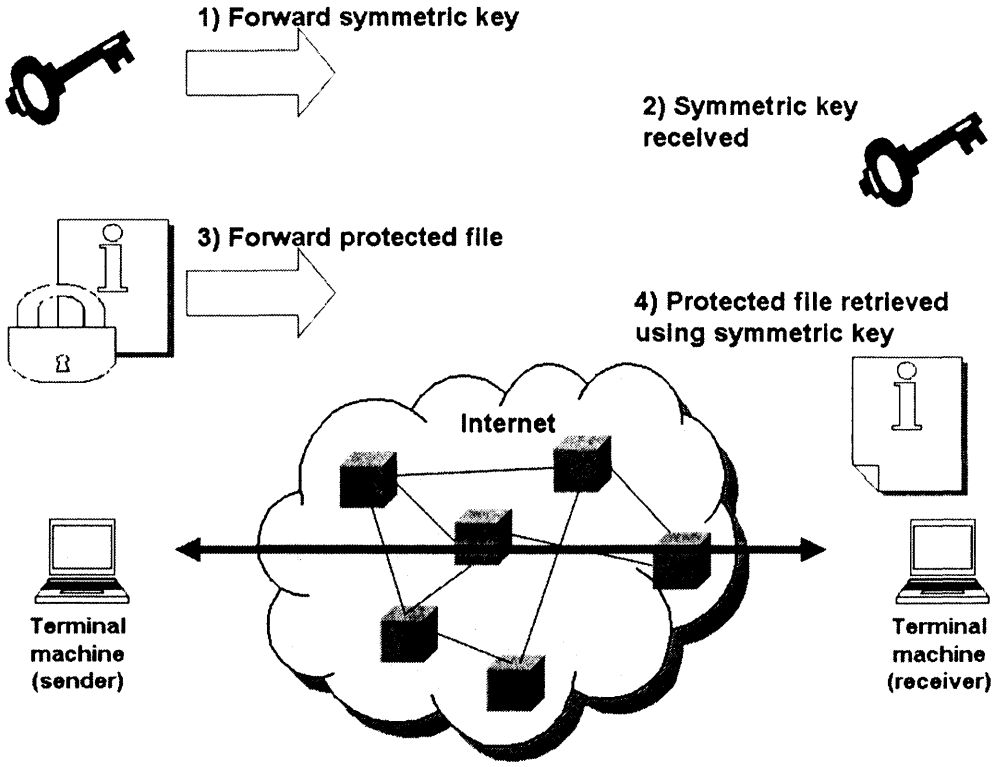

Figure 9 – Secured communications using symmetric keys

Figure 9 shows how symmetric cryptography is used in today's Internet. The sender must first establish with the receiver a symmetric key. This key will be used for protecting future traffic between the two. In a hop-by-hop environment, in order to transmit a packet securely to its next hop, the node (that the packet resides) must share a symmetric key with the packet's next hop. This shared symmetric key is then kept securely on the node, and it will be used for future cryptographic operations on packets transmitted between the two nodes. Note

that the basic assumption when dealing with network security is that the public Internet is insecure. As a result, the use of symmetric cryptography would be successful only if the shared key was distributed and kept securely. As a summary, an efficient, scalable and secure symmetric key distribution mechanism is needed to use symmetric cryptography in today's Internet and active networks.

One may suggest not to bother with dynamic key distribution, but to pre-distribute shared keys to all nodes in the network. Pre-sharing keys might seem simple, but it is a static and a non-scalable key distribution method. It requires manual key generation and distribution. Internet Key Exchange v2 (IKEv2) (section 2.10 on p.55) is a standardised protocol for dynamic key exchange in the Internet. IKE is published as a series of Request For Comments (RFCs). The latest version of IKE is IKEv2 [30], which has replaced its ancestor IKEv1 [31][32][33]. A list of the important differences between IKEv1 and IKEv2 can be found in [34]. The flaws of IKEv1 are identified in [35]. When compared to IKEv1, IKEv2 is flexible but less complex, simplified, and with enhanced security techniques to address network attacks such as DoS attacks [36]. Since IKEv1 is now obsolete, IKEv2 is discussed in this thesis.

2.2 Asymmetric Cryptography for Hop-by-hop Security

Asymmetric cryptography [37] uses two keys (i.e. known as a key pair): one for encryption, the other one for decryption. Asymmetric cryptography is generally used for authentication, but it may also be used for confidentiality protection. A private key is used for digitally signing a piece of data, and the authenticity of the signed data can be verified by using the corresponding public key. For instance, the Digital Signature Standard (DSS) [38][39] uses asymmetric

cryptography for digital signature generation. Asymmetric cryptography can also be used for confidentiality protection, such as the Rivert Shamir Adelman (RSA) algorithms [40]. In RSA, a piece of data is encrypted by using the recipient's public key. Only the recipient can decrypt the encrypted message (because the recipient is the only person who owns the corresponding private key for decryption). Note that asymmetric cryptography is more computationally expensive than symmetric cryptography (section 8.5.3 on p.166).

The advantage of using asymmetric authentication is that source authentication is achieved, and there is no need for dynamic shared symmetric key establishment. Since Public Key Infrastructure (PKI) (section 8.1 on p.162) [41][42] is used to distribute (public) keys, there is no need for each active node to dynamically generate and distribute hop-by-hop (symmetric) keys; but this implies a solution based on asymmetric cryptography would need a scalable and standardised certificate retrieval mechanism i.e. PKI. Because PKI has been deployed widely in a large scale (all web browsers support PKI), it is reasonable to assume that active nodes on fixed networks, i.e. the Internet, have access to PKI. However, asymmetric cryptography has some serious drawbacks when being used for hop-by-hop protection; not only because of the expensive performance overhead incurred, but also for the following reasons:

- Unique private key ownership

Asymmetric cryptography requires a private key for signing, and a public key for verification. A packet will be signed by the source's private key at the source node, and verified by the source's public key at the intermediate node; but as the source's private key is kept locally on the source node, the intermediate node is unable to reproduce the source's signature after modifications to the

packet have been made on an intermediate node.

- Multiple signatures on packets

To solve the previous problem, each node would have to sign the packets by using its own private key. Thus the intermediate node would have to sign the modifications made on the packets with it own private key; but then the old signature, i.e. the source's signature (that was generated by the source), would be overwritten. Thus, the destination node will not be able to verify the packets' source authenticity.

To go round these problems, the packet structure would have to provide more than one field for keeping digital signatures. One field is used for keeping the source's digital signature, another field is used for keeping the digital signature generated by the intermediate node, and so on. Thus, the following packet format should be used (Figure 10). Note that Packet Data is the original data generated by the source node, whereas Packet Data' is the data created by an intermediate node.

Figure 10 - Packet format for asymmetric authentication

However, asymmetric cryptographic operations are much slower than symmetric operations (Question 6 on p.166). Thus, digitally signing each modified part of each packet at each intermediate node would generate an undesirable performance overhead. Furthermore, if each intermediate node adds a new digital signature to the packet, then the size of the packet will grow proportionally as it traverses more and more intermediate nodes.

- Integrity and confidentiality protection

Packet confidentiality can only be protected if the source node encrypts the packets' data with the intermediate node's public key; but since any node that has access to the PKI may obtain the intermediate node's certificate (hence its public key), authentication could not be achieved. This is because any one can encrypt any data with the intermediate node's public key, thus the intermediate node has no way to determine which node had carried out the encryption, unless additional symmetric/asymmetric operations are involved. This could be solved by requiring the source node to encrypt the packet's data with the recipient's public key as before, and subsequently sign the encrypted data with its own private key. However, this arrangement requires additional overhead.

2.3 A Packet Language for Active Networks (PLAN)

The developers of PLAN [22] claim PLAN (which is later on further developed, and known as Safe and Nimble Active Packet (SNAP) [16] to be the first practical active packet protocol. The scope of PLAN was to develop a novel, efficient, and scalable active network protocol. It is not within the scope of this thesis to discuss the applicability of PLAN or SNAP. However, security issues were not addressed in PLAN. The developers of PLAN have only made general suggestions on using cryptographic techniques for active network security, but they did not present the actual design or implementation for active network security. Furthermore, hop-by-hop security aspects were not addressed in PLAN. A security solution for SNAP, developed by the author, can be found in section 2.8 (p.52).

2.4 Secure Active Network Environment (SANE)

Secure Active Network Environment (SANE) [43] provides a set of workarounds for avoiding or minimising the overhead raised by SA negotiations in active

networks, and provisioning was made for key exchange [44]. However, the key exchange approach did not address hop-by-hop SA negotiations, and it is not clear whether the approach has any performance advantages, given that only a set of cryptographic benchmarks were provided

.

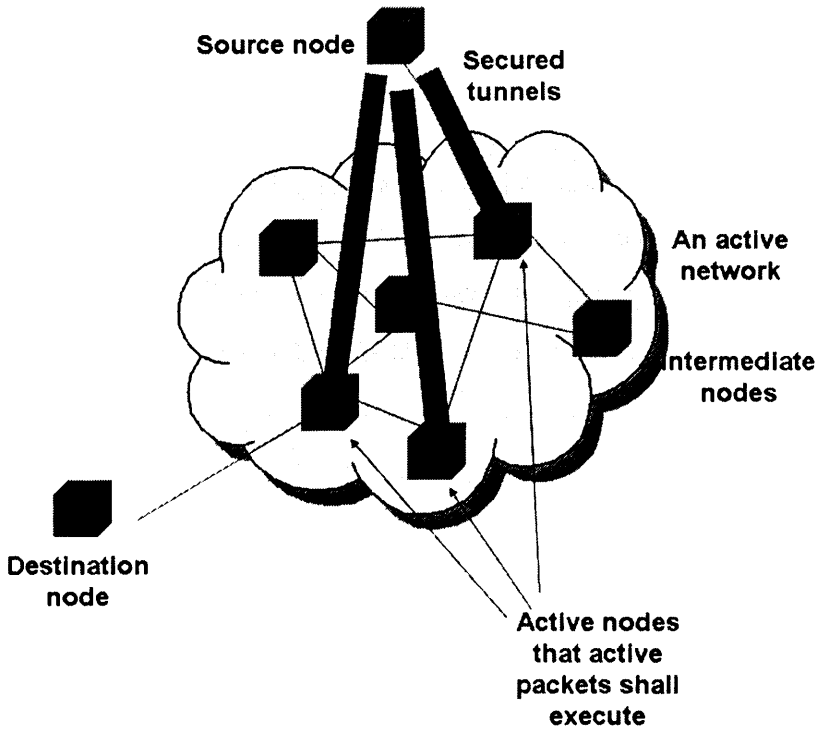

Figure 11 – SANE proposal: individual tunnels

In one workaround, SANE requires the keys to be individually negotiated between the principal and each node (Figure 11). Once SAs have been established between the source node and all intermediate nodes, active packets could be sent directly from the source node to each of the intermediate nodes respectively. This is not scalable as this is a centralised approach: the source node is involved in sending active packets to all other nodes directly. This arrangement also does not support the dynamic nature of active packets.

In order to pass execution result to another node, the execution results on an intermediate node would have to be fed back to the source node through a feedback system, before the source node could send the modified active packet to the packet's next hop. A feedback system for processing every packet is not scalable; this is because a feedback arrangement creates undesirable performance overhead. In other workarounds, SANE requires either that the principal knows the execution path in advance (which contrasts with the dynamic routing capability of active packets), or that peer-to-peer trust already exists. Moreover, the SANE protocol did not address DoS attacks.

2.5 Secure Active Node Transfer System (SANTS)

The developers of Secure Active Node Transfer System (SANTS) [23][28] have proposed a solution by using a combination of asymmetric and symmetric techniques for hop-by-hop authentication for active packets. Digital signatures as well as *Credential References* (section 8.4 on p.164) are used to protect the end-to-end and hop-by-hop authenticity of active packets in SANTS respectively. Credential references bind an object of identity to a claimant's property such as IP address. In SANTS, both active node integrity and link integrity are assumed by enforcing Hashed Message Authentication Code (HMAC) integrity protection between neighbouring nodes. Note that HMAC [45] is a secure hash function that is used for integrity protection (section 8.7 on p.175). A hash function takes a variable size message, and calculates a fixed size message digest. The message digest is sent along with the message to the recipient. Upon receiving the message (and the message digest), the recipient calculates a new message digest of the received message, and compares the two message digests. If the message has not been tampered with during

transmission, the message digests should match. This is integrity protection. It is computationally impossible to generate the same message digest from two different messages for a secure hash function.

In SANTS, ANTS packets are encapsulated into ANEP. To ease reading, ANTS packets that are encapsulated in ANEP are known as ANEP-SANTS packets in this thesis. ANEP-SANTS packets are authenticated in a per-packet and per-hop basis. At the point of packet injection, each ANTS packet is physically split into two parts: static and dynamic part. Static parts are parts in the packet that shall not be modified during packet transmission, thus static parts are digitally signed by the source. The source's digital signature on static code provides data origin authentication. The authenticity and integrity of the dynamic parts are protected by per-hop protection i.e. HMAC-SHA1 (section 8.7.3 on p.177).

The original ANEP format is modified in SANTS: the original ANEP Payload field is separated into a static and a variable area for keeping the static Message Digest 5 (MD5) hash identifier of the active codes and dynamic (i.e. network resource bound) data respectively. A new field is introduced to keep a list of credentials (i.e. X.509 identifiers) to support multiple principal attributes and identifiers to the packet as the packet traverses the network. An Option field is also introduced for keeping digital signatures, which are associated with the credential references. Figure 12 shows the resultant ANEP-SANTS packet header format. An ANEP-SANTS packet therefore – as it traverses the network – carries not just the source's signature, but also a series of identifiers of the modifying nodes that the packet's contents has been modified. ANEP-SANTS packets are authenticated upon successful checks on the embedded source's

signature and credential references in the packets at each of the (execution) nodes along the packet's transmission path.

SANTS's ANEP header	
Credential Field	(list of credentials)
Static Payload	(EE header and data)
Origination Signatures	(each signature covers two previous fields)
Varying Payload	(EE header and data)
Original ANEP Options	(src identifier, destination identifier, integrity checksum)
Hop Integrity	(covers everything)

Figure 12 – The modified ANEP packet format defined in SANTS

The SANTS approach suffers from several drawbacks:

- Inefficiency

The idea of packet splitting is suggested in SANTS so that different authentication techniques can be applied to static and dynamic parts of the packet respectively. However, the developers of SANTS have not discussed how they could efficiently separate the contents of an active packet into static and dynamic parts. In the SANTS approach, it was said that "*The static area of our packets (ANEP-SANTS Static Payload) includes the static portions of the EE (ANTS) header... and the static portions of the data payload. The variable area of our packet (ANEP-SANTS Variable Payload) includes the variable fields of the EE (ANTS) header and the variable portions of the data payload*". SANTS requires *each* ANTS packet to be *physically split* at the sender, and *re-united* at the receiver. The splitting process would involve analysing the contents of an ANTS packet, and deciding on which part(s) is(are) static and which parts is(are) dynamic. Then, each part must be placed accordingly into the ANEP-SANTS packets. Indicators on which parts of the ANEP-SANTS packet refer to static code and dynamic code of the original ANTS packet must be added to the

header of the ANEP-SANTS packet. This information is crucial because the receiver needs to know how to un-marshal (or re-create) the original ANTS packet upon receiving the ANEP-SANTS packet. The performance overhead for keeping track of which-bits-belong-to-where of an ANEP-SANTS packet during the packet splitting and re-uniting processes is not discussed in SANTS. Note that active routers are built on top of conventional routers (no special hardware added), thus active technologies do not increase the physical capability of a conventional router (i.e. processing speed), but only the functionalities of conventional routers are enhanced through active technologies. Note that conventional routers are designed to pass packets forward, *not* to process packets. Thus adding additional functionalities to conventional routers is already adding extra load to the nodes. It would be undesirable to introduce any *unnecessary* processes in order to process active packets. An active network hop-by-hop security solution, i.e. the FAIN ANEP-SNAP Packet Engine, proposed by the author of this thesis suggests that no packet splitting is actually required (section 2.8 on p.52).

- Packet format modification

SANTS requires modifications to the original ANEP packet format (splitting the ANEP payload into two parts). Although the ANEP packet format has not been used in practise due to the limited applicability of active technologies on real networks (i.e. publicly accessible networks other than testbeds), *unnecessary* modifications to existing standards should be limited for inter-operability. As it will be discussed in section 2.8 (p.52), packet splitting is not needed anyway. Thus, this modification to ANEP packet format is unnecessary.

- Lack of confidentially protection

Another drawback of the SANTS approach is that, with message digest (i.e. HMAC), only partial authentication and integrity protection are provided, but no confidentiality protection to the overall packet's contents is provided. As shown in Figure 12 (p.45), both the static code and dynamic data of an active packet are transmitted in the form of clear text (no field in the packet is encrypted). The developers of SANTS did not identify the need for confidentiality protection for active packets in their paper; and the SANTS approach has made no provisioning for confidentiality protection. Currently, active networking has not been deployed on a practical network such as the Internet. Thus, the issues arising from the actual practical usage of active technologies are largely unknown. Therefore, one may argue whether the confidentiality of active packets should be protected. However, experiments have shown that the flexibility of active technologies allows active packets to carry *control* code for specific service deployment [17][46][47][48], or for node states information query [16]. The control code may contain specific node operational status information, for example: "if software component w is currently running on this node, execute this code only when the flow on interface_x exceeds y bytes over z seconds; else if...". Specific information on node operational status (in this example the operational status of specific software components, interface names, and packet flow conditions) are potentially sensitive, therefore strong protection to active packets, i.e. confidentiality protection, is therefore desirable.

- Lack of per-hop key distribution mechanism

The SANTS developers suggested using HMAC-SHA for per-hop protection, but they have not discussed how symmetric keys can be distributed efficiently in a hop-by-hop manner. Later on in another document [27], they suggested using

Sign Key Transport (SKT) as a way for key distribution for per-hop protection. However, SKT has severe drawbacks as a key distribution technique (section 2.6 on p.48).

2.6 Signed Key Transport (SKT)

SKT was proposed by the SANTS developers to support SANTS operations. SKT is a technique to distribute symmetric keys between hops. A summary of SKT is given below.

Recent research [27] suggests that instead of using the full-scale Multiple IPSec approach, SKT could be used for distributing symmetric key in active networks. The idea is that the source generates a symmetric key (i.e. a key to be used for hop-by-hop packet protection), and signs this key with its private key. The signed symmetric key will be sent together with the active packet along the transmission path. Each of the intermediate active nodes will obtain a copy of this symmetric key after a successful check on the source's signature. The signed symmetric key is only distributed once, and kept locally on each node. The symmetric key is used to protect the dynamic data of active packets that traverse the (active) network.

The problem is that there is no confidentiality protection for the symmetric key, and hence the confidentiality and integrity of the dynamic data of subsequent active packets are not protected. Any intermediate node, which has intercepted the packet and has access to the source's public key certificate can obtain the symmetric key; and hence any subsequent data encrypted by that symmetric key can be decrypted by any of these nodes. In fact, the developers of SKT admit that a fundamental requirement of their solution is that "*...the packet distributing the symmetric key routes itself to only trusted nodes*". This

assumption cannot be justified in an insecure network such as the Internet. The solution has no provisioning for Man-in-the-Middle (section 8.5.4 on p.167) or replay attacks (section 8.13.2 on p.183). Although it is arguable whether active packets protected by the symmetric key require confidentiality protection, this approach would not fit in situations where strong security is needed since active packets may be used to carry control code.

Furthermore, SKT requires all nodes to share the same key. However, when more than one-pair of hops share the same key, authentication would fail. This is because a mis-behaving member (which owns the shared key) can use the shared key for encryption, but claims that the encryption was carried out by others. There is no way to tell who actually performed the encryption. This is similar to the situation in which there is a lock, but more than one people have a copy of the key. Thus, when the lock is un-locked, there is no way to tell who actually unlocked the lock. So, unless the symmetric key is replaced with a new symmetric key at each hop, per-hop authenticity and anti-replay are not protected. However, these issues were not addressed in SKT. Another drawback of the proposed SKT is its flexibility: there is no SA negotiation. SA negotiation is the process through which two or more nodes can negotiate the cryptographic parameters to be used. For instance, the source node may support a particular cryptographic algorithm, whereas the recipient node does not. In this case, the source node cannot use that cryptographic algorithm (because the recipient would not be able to process packets subjected to an unknown cryptographic process). SA negotiation is therefore essential in order to ensure that both parties are capable of processing cryptographically processed packets. Without support for SA negotiation, the source simply

creates a symmetric key and distributes the key along with the packet. The source would expect intermediate hops to accept what was created/chosen by the source (i.e. keying materials, the cryptographic algorithm used to generate the key). As explained in an earlier section, this would not work if the recipient did not support the chosen cryptographic algorithm. With no SA negotiating power, SKT would only be suitable for a small-scale active network, in which all active nodes involved are pre-ensured that they support a pre-chosen set of cryptographic algorithms/materials.

2.7 Centralised Keying Server (KSV)

Another solution for active packet authentication and integrity protection was proposed by Krishnaswamy in [49]. It was proposed that a centralised Keying Server (KSV) should be used for dynamic SA setup across a set of active nodes (Figure 13). A Key Management Module (KMM) is installed on each of the participating active nodes. The KMM is implemented as an extension of IKE. The KMM handles all interactions between the active node itself and the KSV. Each participating active node must first register itself with one KSV. Registration is done when the active node boots up. Keys are then established per active node. Key establishment between active nodes is done through IKE, and IPSec SAs are established between active nodes consequently. Thus, the KSV maintains a list of keys that it has established with the participating active nodes. The established keys are then used for IPSec protection between the nodes. Note that this arrangement therefore protects the hop-by-hop authenticity and integrity, as well as addressing SA establishment issues.

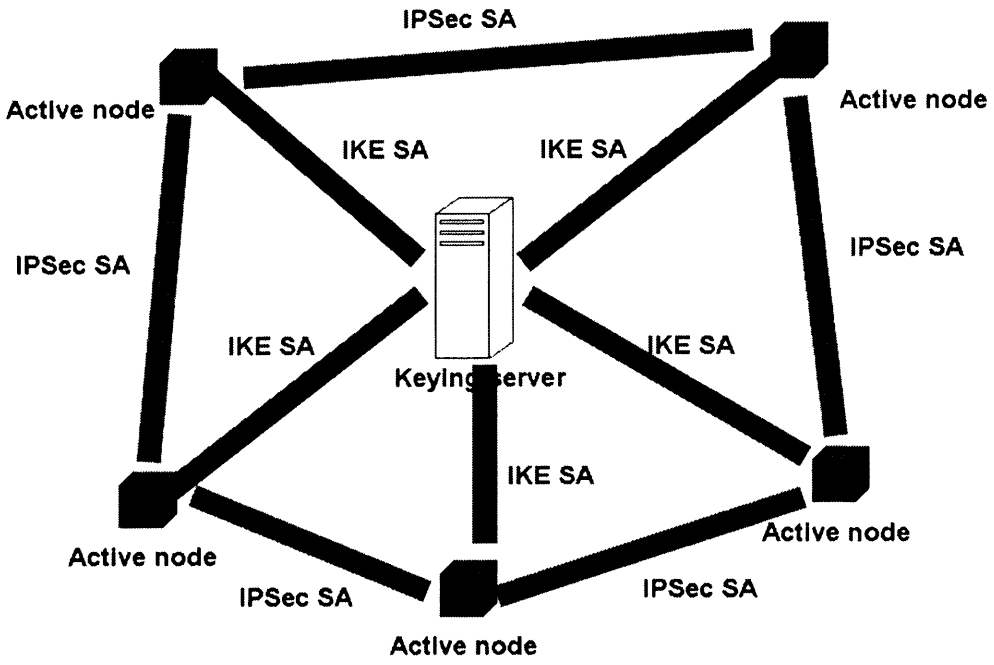

Figure 13 – The KSV model

The proposed KSV solution limits itself to registration with one KSV i.e. a *centralised* approach. This means the proposed solution is limited to a small-scale active network testbed only. The developers of KSV suggested that an active node might register itself with more than one KSV to reduce the effect of centralisation. However, the developers of KSV had not identified how the potential conflict caused by duplicated registration (with more than one KSV) is resolved.

The use of a centralised Keying Server causes scalability problems. If the keying material for one of the participating nodes is updated, the KSV must inform *all* participating nodes regarding this update, the participating nodes must validate and acknowledge the update message, and the KSV must validate the integrity and authenticity of the acknowledgement message... and

so on. This could introduce unnecessary performance overhead. A distributed solution would be to enable key establishment and maintenance to the participating active nodes, instead of dedicating an entity (i.e. the KSV) to carry out these jobs.

KSV uses IKE for SA negotiations. IKE was designed to establish SA between two static end points. The performance overhead of using IKE for per-hop SA negotiation was not addressed in KSV.

2.8 FAIN: ANEP-SNAP Packet Engine

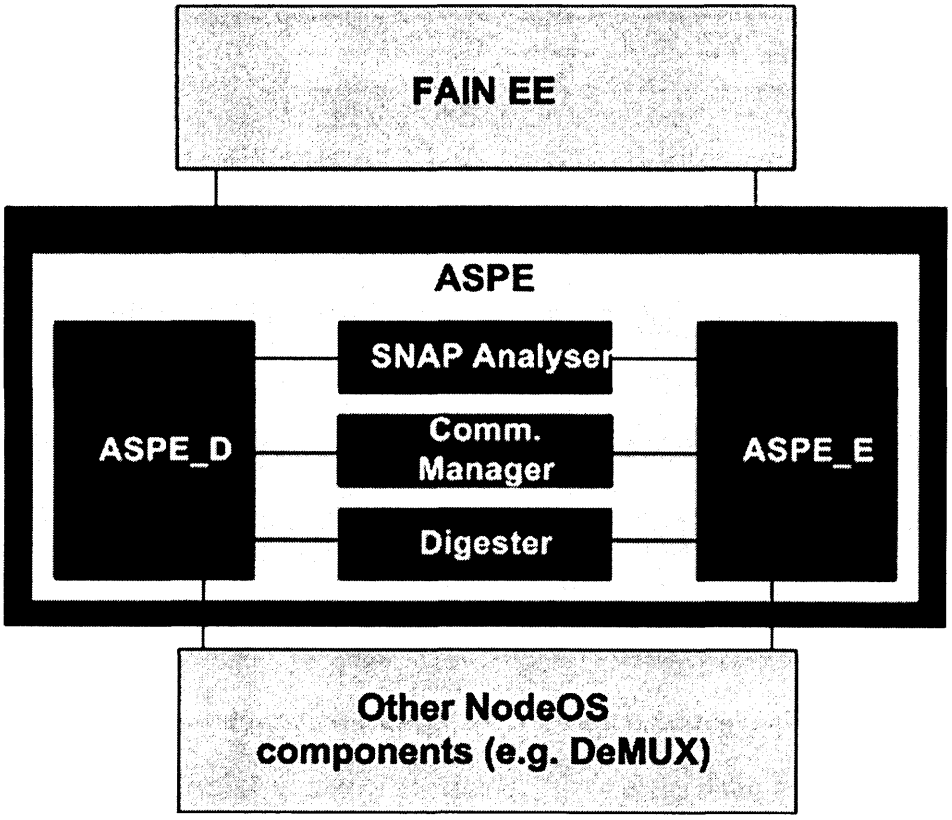

Figure 14 – The components of ASPE

The FAIN solution, known as the ANEP-SNAP Packet Engine (ASPE), was proposed and developed by the author of this thesis in the IST-FAIN project [13]. The FAIN solution suggests that no physical packet splitting or reuniting is

required. Essentially, the ASPE is part of the Security Component of the FAIN NodeOS. It encrypts and decrypts SNAP packets used in the active networks. The packet (de)multiplexing component on the FAIN NodeOS delivers encrypted ANEP-SNAP packets (that are captured on the wire) to the ASPE. The ASPE decrypts the packets, and sends the packets to their desired EEs. The ASPE is also responsible for encrypting (SNAP) active packets before they are sent to the networks through the NodeOS.

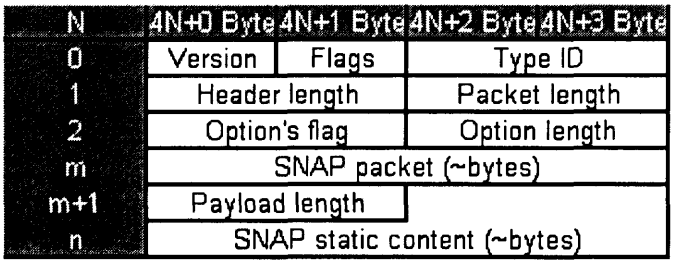

Figure 15 – FAIN ANEP-SNAP packet format

Figure 15 shows the FAIN ANEP-SNAP packet format. The entire SNAP packet is placed in the payload field of the ANEP packet format. The static code of the SNAP packet is appended to the packet. Static parts of the SNAP packet are digitally signed at the source node. This digital signature is static whilst the packet is traversing the network. The hop-by-hop authenticity and integrity of the *entire* packet (which includes *both* static and dynamic parts) are protected by using IPSec Authentication Header (AH). At each hop, after security checks, the entire active packet is simply extracted. There is no need to physically split and re-unite active packets. Although under this arrangement, static parts of the packet will be subjected to both the asymmetric (being signed by the source) and symmetric cryptographic processes (per-hop protection at each hop); the advantage is that there is no need to physically split and re-unite active packets at each hop. Since there is no need to keep the dynamic data separately, this

approach requires only one Payload field (that is used to keep the entire active packet). Thus, there is no absolute need to modify the ANEP format. To enforce hop-by-hop protection, each pair of hops shares a different IPSec SA. This is known as *multiple IPSec*. This solution, however, addresses neither the confidentiality, nor the hop-by-hop key distribution issues. To enforce confidentiality protection, IPSec Encapsulated Security Payload (ESP) is one of the potential solutions (section 2.9 on p.55).

Question 3: Instead of equipping each pair of nodes with different symmetric keys, i.e. multiple IPSec, is it possible to equip all nodes in the network (or an administrative domain) using the same symmetric key i.e. use *multicast IPSec*? In multiple IPSec (AH), the symmetric key is used to protect the authenticity and integrity of an active packet. Upon receiving the protected packet, the receiver looks up information (from the corresponding IPSec SA) to determine the corresponding symmetric key in order to process the packet. The receiver expects the source host to be the only other host that knows the secret key. As a result, the packet's source authenticity is verified. The important point to note is that only two nodes own the same symmetric key, so non-repudiation protection on the packet is enforced. Enforcing non-repudiation protection on the static code of active packets is important in active networks, this is because control code is potentially damaging. With non-repudiation, the actual creator of the static code cannot deny of any wrong doings should the control code cause any damage to the networks.

In *multicast* IPSec, however, all hosts share the *same* key. Source authentication and non-repudiation protection based on shared key is not possible when more than one pair of hosts share the same key. All one can

determine from multicast IPSec is that the data is encrypted by a valid key by a member of the IPSec multicast group. However, the actual identify of the individual node who encrypted the data is not revealed.

As discussed in an earlier section, currently, the issues arising from the actual practical usage of active technologies are largely unknown; so potentially, active packets can be multicast to desired node groups. It should be noted that (as explained in the last paragraph) multicast security is very different from unicast security. This thesis does not discuss the use of multicast IPSec or multicast security approaches for active networks. For more background on secure IP multicast, readers are referred to [50]. For detail of multicast IPSec and challenges of key exchange for multicast IPSec, readers should refer to [51][52][53].

2.9 IPSec ESP

To solve the described per-hop authentication and confidentiality protection problem, and the lack of SA negotiation power in SKT and its variations, IPSec ESP [54] could be used between hosts. Unlike SKT, IPSec uses IKE, which supports SA negotiation; thus adds creditability when deployed over a heterogeneous network such as the Internet. More specifically, IPSec ESP provides authentication, integrity and confidentiality protection. If IPSec ESP is applied, the entire active packet is placed in the Protected Payload. In this case, the confidentiality of the entire active packet is protected. IPSec ESP, however, still does not address hop-by-hop SA establishment.

2.10 Internet Key Exchange v2 (IKEv2)

2.10.1 An Overview

IPSec uses IKE for SA establishment. This section discusses the use of IKEv2

for hop-by-hop security.

IKE is an automated key management protocol used by IPSec. The key feature of IKE is that it allows two communicating peers to negotiate SA parameters before establishing IKE and IPSec SAs. It is the provisioning of SA negotiation of IKE that adds credibility to IKE in terms of its flexibility and scalability. IKE uses Diffie-Hellman (D-H) Key Exchange (section 8.6 on p. 168) for secure private symmetric key establishment between two peers. D-H key exchange is a protocol that enables two peers to establish a shared secret (key) instantly over an insecure link without having to transmit or disclose any secret information over the insecure link, or using any pre-configured or pre-distributed parameters. As such, IKE is not only flexible and scalable, it is also secure.

The latest version of IKE is IKEv2. The benefit of using IKEv2 (compared to SKT and its variants) is that IKEv2 retains SA negotiation power and the power to establish shared secrets securely between peers (through D-H key exchange). It should be noted that, according to the IKEv2 RFC, IKEv2 must first establish an IKEv2 SA between two peers, then the established IKEv2 SA is used to establish another (say, IPSec) SA between the same pair of peers, prior to the secure transmission of a (active) packet using the subsequently established SA. The purpose of this (rather redundant) arrangement was so that IKEv2 could be used as a generic key exchange protocol, that could be used as the underlying key exchange protocol on top of which other applications (say, IPSec) might establish their own SAs. However, because of the hop-by-hop nature in active networks, it is essential to investigate how to reduce the performance overhead incurred by per-hop IKEv2 and IPSec SA negotiation processes, and to remove any redundancy. Furthermore, as it will

be discussed later, IKEv2 is subjected to DoS attack (section 4.6 on p.110). Essentially, IKEv2 can be deployed with or without Perfect Forward Secrecy (PFS). PFS is a property in key exchange protocol that enables strong security. However, enforcing PFS in key exchange incurs a high overhead. Discussion of PFS and its performance is out of scope of this thesis, readers are referred to the Appendix for more detail (section 8.12 on p.180). The basic IKEv2 does not support PFS, whereas a variant of IKEv2 supports PFS.

2.10.2 Key Exchange Process in IKEv2

An IKEv2 key exchange is conducted between two peers: an Initiator (I) and a Responder (R). An Initiator is a node that starts the key exchange, whereas a Responder is a node that is responding to the key exchange request initiated by the Initiator. The basic IKEv2 protocol (no PFS) involves an exchange of four messages to complete a key exchange (Figure 16). Note that, in the following sections, items in square brackets are optional, whereas the authenticity, integrity, and confidentiality of the items in curly brackets are protected by a shared symmetric key set (see shortly later).

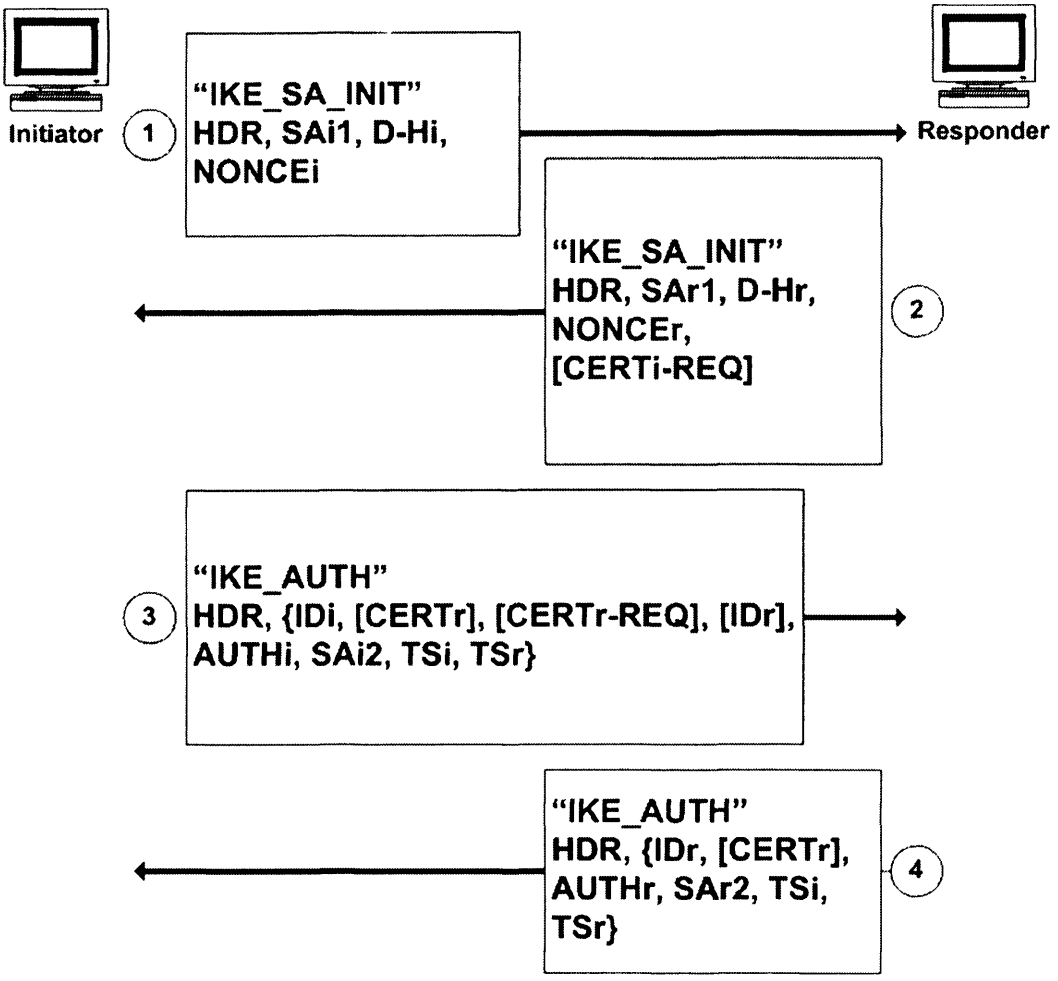

Figure 16 – IKEv2 (with no PFS)

In the first message, the Initiator sends to the Responder a request message that contains a header (HDR), its SA (SAi1) which contains a list of preferred or supported security parameters such as encryption algorithms and keysize. D-Hi is the D-H public values generated by the Initiator for this key exchange, and NONCEi is a 128-bit randomly generated nonce. These values (together with the D-H public values and the nonce generated by the Responder) are used later on in the IKEv2 protocol to generate a shared secret between the Initiator and the Responder. Readers are referred to the Appendix for detailed information on the D-H algorithm (section 8.6 on p. 168). The Responder

intercepts the first message, and responds with the second message, which contains the Responders SA (SAr1), its D-H public values (D-Hr), a randomly generated nonce (NONCEr), and optionally a request for the Initiator's PKI certificate ([CERT-REQ]). SAr1 contains the Responder's choice of the Initiator's SA. SAi1 and SAr1 are stored locally on the Initiator and the Responder respectively for future reference. A SA is identified on the node by a Security Parameter Index (SPI).

Once the Initiator and the Responder have exchanged the first and the second message, both peers are capable of generating a shared secret (g^{xy}) using the D-H algorithm. Subsequently, a shared secret key seed (SKEYSEED) can be generated (Equation 1), from which a set of shared keys (SK_a, SK_e, SK_d and SK_p) can be derived (Equation 2). A shared secret key seed, i.e. SKEYSEED, is established as follow by using concatenation (section 8.2, p.164):

$$SKEYSEED = PRF(NONCEi \mid NONCEr, g^{xy})$$

Where

| = the notation of concatenation

Equation 1

When generating the SKEYSEED, nonces are applied to the D-H shared secret with the use of Pseudo-Random Function (PRF) (section 8.13 on p.182). The purpose is to add randomness to the resultant shared secret (by applying the randomly generated, never re-used nonces), and to ensure that the resultant shared secret key seed (SKEYSEED) has a standard size. The subsequent shared key set is generated as follow:

$$SK\_d \mid SK\_a_i \mid SK\_a_r \mid SK\_e_i \mid SK\_e_r \mid SK\_p_i \mid SK\_p_r = PRF+(SKEYSEED, NONCEi \mid NONCEr \mid SPIi \mid SPIr)$$

Equation 2

SPIi and SPIr are the SPIs of the SAs that are stored locally on the Initiator and

the Responder respectively. Again, the idea of applying the PRF function is to ensure that randomness and standard size keys (of the shared key set). Equation 2 shows that the SK_e, SK_a, SK_d and SK_p keys are generated from the output of PRF+ (section 8.13 (p.182). According to Equation 18 (p.183), SK_d is determined as follow:

$$SK\_d = PRF(SKEYSEED, NONCE_i \mid NONCE_r \mid SPI_i \mid SPI_r \mid 0x01)$$

Equation 3

$$SK\_a_i = PRF(SKEYSEED, SK\_d \mid NONCE_i \mid NONCE_r \mid SPI_i \mid SPI_r \mid 0x02)$$

Equation 4

$$SK\_a_r = PRF(SKEYSEED, SK\_a_i \mid NONCE_i \mid NONCE_r \mid SPI_i \mid SPI_r \mid 0x03)$$

Equation 5

$$SK\_e_i = PRF(SKEYSEED, SK\_a_r \mid NONCE_i \mid NONCE_r \mid SPI_i \mid SPI_r \mid 0x04)$$

Equation 6

$$SK\_e_r = PRF(SKEYSEED, SK\_e_i \mid NONCE_i \mid NONCE_r \mid SPI_i \mid SPI_r \mid 0x05)$$

Equation 7

$$SK\_p_i = PRF(SKEYSEED, SK\_e_r \mid NONCE_i \mid NONCE_r \mid SPI_i \mid SPI_r \mid 0x06)$$

Equation 8

$$SK\_p_r = PRF(SKEYSEED, SK\_p_i \mid NONCE_i \mid NONCE_r \mid SPI_i \mid SPI_r \mid 0x07)$$

Equation 9

Note that all subsequent messages exchanged between the Initiator and the Responder will be protected by the following keys:

- SK_e (SK_e_i, SK_e_r)

This key is for encryption. There is one SK_e key for each direction, but each peer owns both SK_e_i and SK_e_r keys.

- SK_a (SK_a_i, SK_a_r)

This key is for authentication and integrity protection. There is one SK_a key for each direction, but each peer owns both SK_a_i and SK_a_r keys.

- SK_d

This key is used to derive subsequent keys for CHILD_SAs. There is only *one*

SK_d key.

- SK_p (SK_p_i, SK_p_r)

This key is used to generate the AUTH Payload. There is one SK_p key for each direction.

The third message in the IKEv2 protocol is sent by the Initiator to the Responder. The message contains an encrypted and integrity payload which contains identity information of the Responder (IDi), the Initiator's PKI certificate (optional), a request for the Responder's PKI certificate (optional), a specified identity of the Responder that the Initiator wishes to communicate with (optional, in case the Responder hosts multiple identities and the Initiator knows in advance which of the Responder's identities that it wants to establish a SA with), some authentication data (AUTHi), a SA which contains additional security parameters that the Initiator wishes to negotiate with the Responder (SAi2, that is needed to establish an IPSec SA), and some Traffic Selectors (TSi and TSr, which specifies the set of port numbers that applications are allowed to use for the established SAs). The AUTHi is a digital signature created by using either the Initiator's private PKI key, or a pre-shared secret shared between the Initiator and the Responder[7], in order to provide authentication and integrity protection of the *first* message in the key exchange. AUTHi covers the following:

HDR, SAi1, D-Hi, NONCEi, NONCEr, PRF(SK_pi, IDi)

Figure 17 – Items covered in AUTHi

Note that the Initiator countersigns the Responder's nonce (NONCEr) in AUTHi.

[7] The use of pre-shared secret is not considered in this thesis: this is because a key exchange protocol is to establish a shared secret. The assumption of having a pre-shared secret to establish a secret is a chicken-and-egg problem.

The idea of countersigning nonces by peers is to prevent impersonation attacks (section 4.2 on p.100). IDi is PRF with SK_pi in order to added security, as well as to ensure that the ID of the Initiator is of a standard size. The SK_pi key is freshly created for this session, so that each AUTHi from the same Initiator (and of the same ID) actually covers the same ID but of different contents. Note that a checksum that covers the entire message 3 is created by using the SK_a key. The checksum is appended to the message for authenticity and integrity protection.

Once the Responder has received message 3, the Responder verifies the protected payload in message 3 by using its own set of shared keys. SK_a is used to verify the integrity of the message, SK_e is used to decrypt the encrypted payload. If these verifications are successful, message 3 is verified. The authenticity and integrity of message 1 is then verified by verifying AUTHi. If this verification is successful, message 1 (from the Initiator) is verified. In this case, the Responder sends to the Initiator the fourth (and the last) message. This message contains again a protected payload which contains the Responder's ID (IDr), the Responder's PKI certificate (CERTr) (optional), the Responder's choices on SAi2, TSi and TSr. Note that a digital signature (AUTHr) that is created by the Responder is also included. AUTHr is used to provide authenticity and integrity protection to message 2. AUTHr covers the following items:

HDR, SAr1, D-Hr, NONCEr, NONCEi, PRF(SK_pr, IDr)

Figure 18 – Items signed by AUTHr

Again, the Responder countersigns the Initiator's nonce to prevent impersonation attacks. All other items are signed for the same purpose as

above except that this time, the Responder is protecting the contents of message 2, instead of the contents of message 1.

2.10.3 Enforcing PFS Support in IKEv2

Note that the basic IKEv2 does not provide PFS support. In order to provide PFS support, another set of message exchanges is required to exchange another set of D-H public values. This extra set of message exchanges consists of two messages being exchanged. In other words, in order to enforce PFS, the total number of message exchanges is now six instead of four. The following are the two extra messages that are required to enforce PFS:

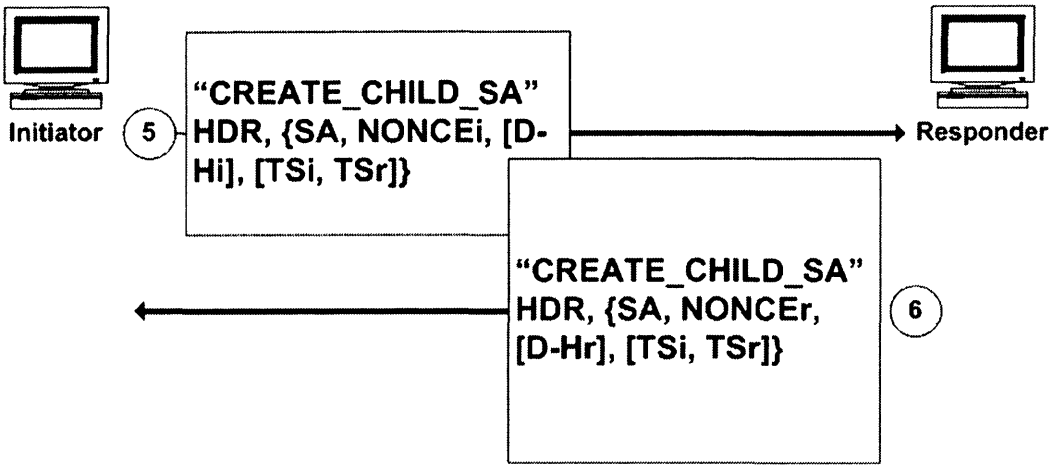

Figure 19 – Extra exchanges for establishing CREATE_CHILD_SA

All the terms are self-explanatory; but it should be noted that in order to enable PFS, another set of D-H public values must be exchanged. In brief, two cycles of shared key set establishment are required. As explained in section 8.12 on p.180, this is an expensive process in terms of performance.

This thesis proposes that, using some useful concepts of IKEv2, but omitting the drawbacks of IKEv2, with new message contents and sequence, a much simplified SA negotiation sequence with much less performance overhead can

be used in order to transmit active packets *during* SA negotiation instead of after. The established SA can be retained to protect subsequent packets being transmitted across the same link.

Question 4: Is it possible to reuse the established hop-by-hop SAs for protecting both active and passive packets, so that the undesirable performance can be absorbed?

Security for passive packets is applied end-to-end. A passive packet is encrypted at source, and only decrypted at the recipient. Intermediate nodes simply forward passive packets. Thus, for passive packets, SA is established end-to-end.

Since SAs for active packets are established in a per-hop fashion, the resultant hop-by-hop keys that are kept locally on intermediate nodes are of no use to the user data which are usually protected end-to-end (unless the destination of passive packets is the next hop). Thus, generally, the end-to-end SAs established for passive packets cannot be re-used for active packets, unless the active packets are transmitted between two end points only.

This further implies that the "re-usability" of the locally kept symmetric keys on intermediate nodes is limited to protecting subsequent (active) packets travelling across the same pair of hosts.

2.10.4 Use of COOKIEs for Addressing DoS Attacks

Another variant of IKEv2 uses *COOKIE* (section 8.3 on p.164). As indicated in the IKEv2 RFC: "*... an endpoint could use cookies to implement limited DoS protection*" [55]. One common type of DoS attacks identified in the IKEv2 RFC would be that a Responder was sent large number of initialisation messages from a DoS attacker; the Responder would then be driven to handle many

initialisation messages, which eventually caused CPU exhaustion on the Responder. Furthermore, the IKEv2 developers claimed that this type of DoS attack could be originated from legitimate IP addresses, but other than the IP address of the node that the attack was actually originated [56]. In this case, the Responder's legitimate reply messages would be sent to other victims (i.e. the nodes that actually own those legitimate IP addresses). Thus, this would create another wave of DoS attacks in the network. The IKEv2 developers claimed that by using COOKIEs in the key exchange protocol, it would be possible to ensure that the Initiator is indeed sending off initialisation request message from the IP address that it claims it owns. However, as it will be described in section 4.6 (p.110) and section 5.5 (p.150), the use of COOKIE in IKEv2 does not provide a complete protection against DoS attacks.

The basic idea is that when an IKEv2 Responder receives an initial request from the Initiator, the Responder does not respond through the normal procedure (as specified in Figure 16), but responds with an empty message that contains a COOKIE. A COOKIE is a randomly generated number that is created by this equation (Equation 10):

$$\text{COOKIE} = \langle \text{VersionIDofSecret} \rangle \mid \text{Hash}(\text{NONCEi} \mid \text{IPi} \mid \text{SPIi} \mid \langle \text{secret} \rangle)$$

Equation 10

The idea is that the Initiator, upon receiving the COOKIE from the Responder, must re-send its original initialisation message with the COOKIE. The developers of IKEv2 claimed that under this arrangement, a DoS attacker that claimed a legitimate IP address (but the claimed IP address was actually owned by another node) would be detected, because the DoS attacker would not be able to get the COOKIE (note that the COOKIE was sent by the Responder to the victim directly), and would not be able to re-send the initialisation request

message with the valid COOKIE. The protocol exchange is almost exactly the same as the basic IKEv2, but with six messages being exchanged. Figure 20 shows the IKEv2 key exchange with COOKIE.

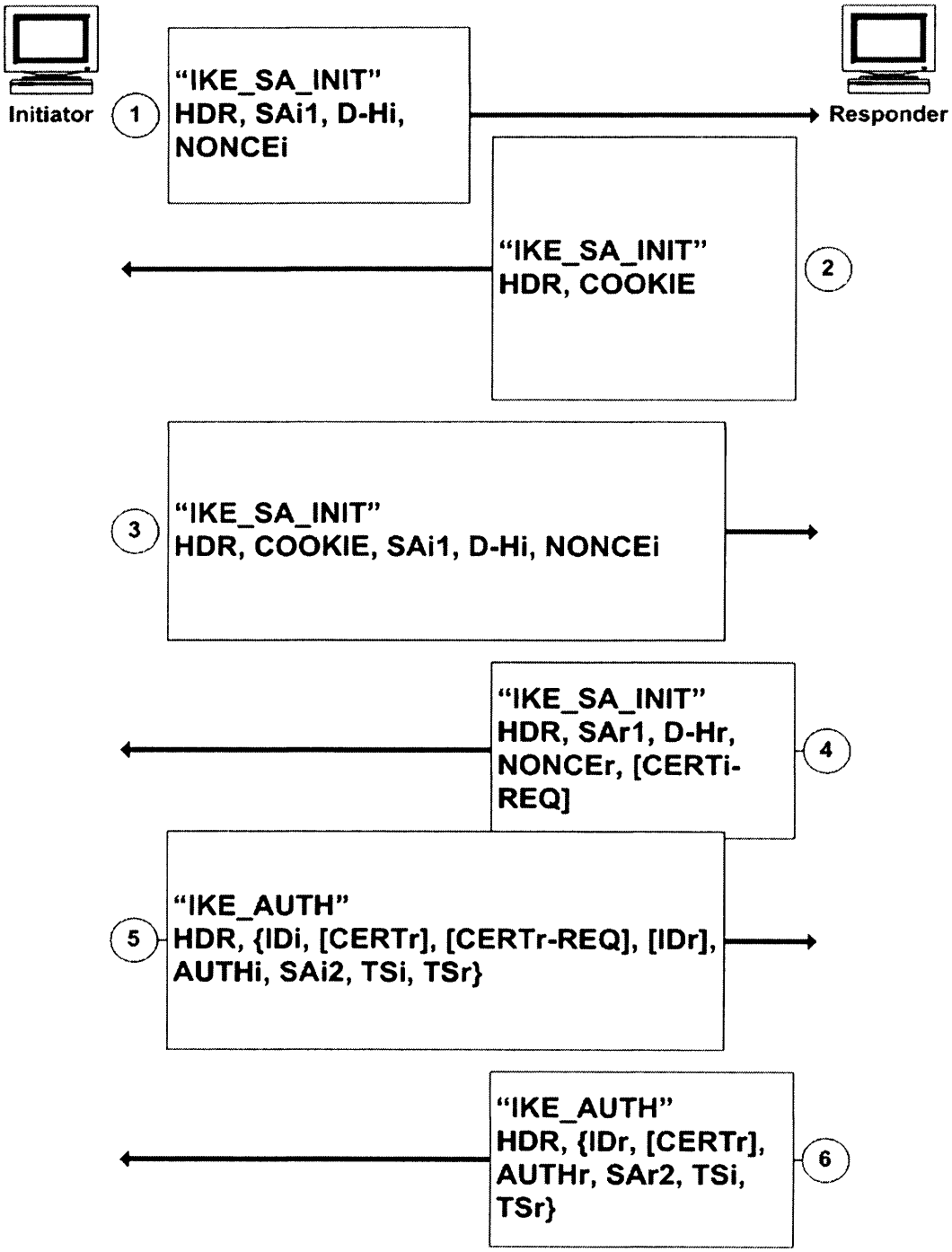

Figure 20 – IKEv2 with COOKIE

2.10.5 Sequence Numbers as Message ID

In IKEv2, each pair of request and reply messages uses the same message ID. A re-transmitted message would use the same message ID that was used in the previous, original message. The message ID is a 32-bit number. The first request message from a peer (i.e. the Initiator) should always use zero as its message ID to start with, and increments sequentially for subsequent messages. For instance, the first IKEv2 message exchange pair would use zero as their message ID, then the second IKEv2 message exchange pair would use "1" as their message ID, and so on.

A peer should maintain two "current" message IDs: one message ID to be used for the next request to be initiated by the peer, and another message ID that the peer expects to see in a request originated from other peers.

This field is cryptographically protected to prevent replay attacks. If after too many message exchanges, the message ID becomes too large to fit in the 32-bit field, the IKEv2 SA must be re-newed. Message IDs are re-set when an IKEv2 SA is re-keyed.

2.11 Just Fast Keying (JFK)

Just-Fast Keying (JFK) [57] has two variants that were designed for different purposes. Each variant consists of an exchange of four messages to complete the key exchange. The developers of JFK claimed JFK to be a DoS-resistant protocol. However, as it will be discussed in section 4.6 (p.110) and section 5.5 (p.150), JFK did not achieve its goal. Furthermore, JFK did not provide identity protection. JFKi was designed to protect the Initiator's identity; whereas JFKr was designed to protect the Responder's identity. It should be noted that neither of the variants protect identity of *both* peers. Figure 21 shows the JFKi protocol:

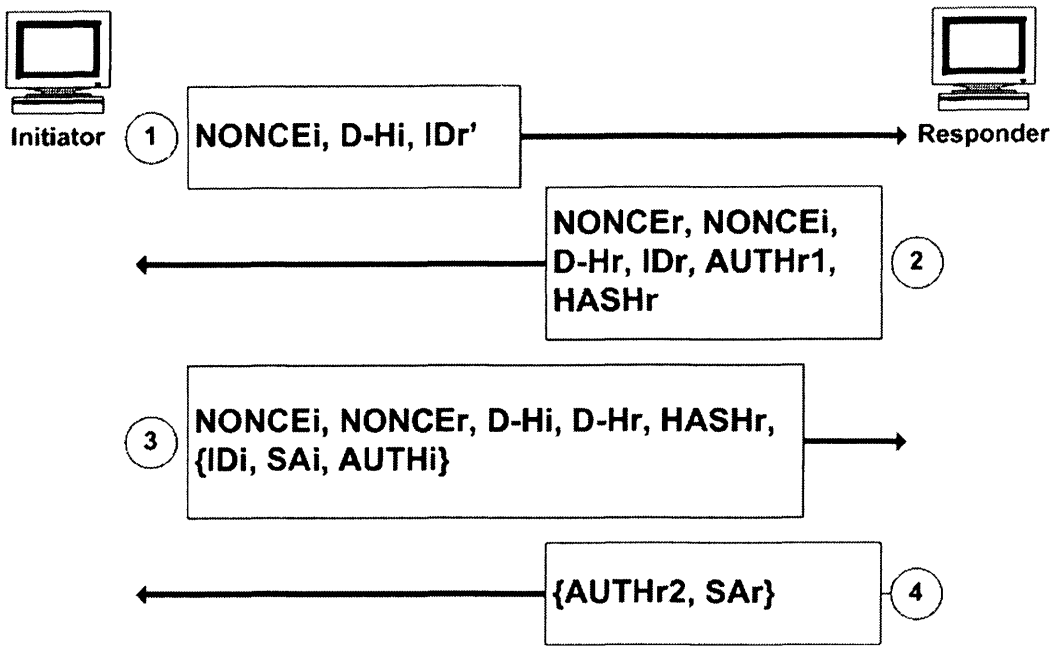

Figure 21 – The JFKi protocol

The first message contains the Initiator's nonce (NONCEi), its D-H public values (D-Hi), and some indictors from the Initiator to the Responder on what authentication data the Responder should be using during the protocol exchange (IDr'). The second message contains the Responder's nonce (NONCEr), the Initiator's nonce (NONCEi), the Responder's D-H public values (D-Hr), the Responder's ID (IDr), a signature (AUTHr1) created by using the Responder's PKI private key, and a hash value (HASHr) that is computed using the Responder's pre-established symmetric key.

AUTHr1: D-Hr HASHr: D-Hr, NONCEr, NONCEi, IPi

Figure 22 – Items covered by AUTHr1 and HASHr

Figure 22 shows the list of items that are covered by AUTHr1 and HASHr respectively. Note that IPi is the IP address of the Initiator. HASHr is computed using the Responder's pre-established symmetric key. Note further that, the

Responder's identity (IDr) is not protected in message 2 (i.e. the authenticity, integrity and confidentiality of the Responder's identity) is not protected.

Message 3 contains the Initiator's nonce (NONCEi), the Responder's nonce (NONCEr), the D-H public values of the Initiator and the Responder respectively, the hash that was created by the Responder, and a protected payload with contains the Initiator's ID (IDi), the Initiator's SA (SAi), and the Initiator's digital signature that is created by using the Initiator's private key over a list of items (Figure 23a). Message 4 is protected, and contains the Responder's signature (AUTHr2) that covers another list of items (Figure 23b), and its replies to the Initiator's SA (SAr).

AUTHi: NONCEi, NONCEr, D-Hi, D-Hr, IDr, SAi	(a)
AUTHr2 : NONCEi, NONCEr, D-Hr, D-Hi, IDi, SAi, SAr	(b)

Figure 23 – Items covered by AUTHi and AUTHr2

JFKr is almost identical to JFKi except that in JFKr, the Responder's identity is protected but not the Initiator. The JFK developers claimed the variants were designed to suit different environments: for example when the identity of the Initiator should be protected, then JFKi should be used (e.g. when initiating a peer-to-peer session with someone who the Initiator does not know); whereas JFKr should be used when the identity of the Responder should be protected (a secure server that acts as a Responder to remote calls generated by unknown Initiators on the network). Figure 24 shows the JFKr protocol, which is self-explanatory.

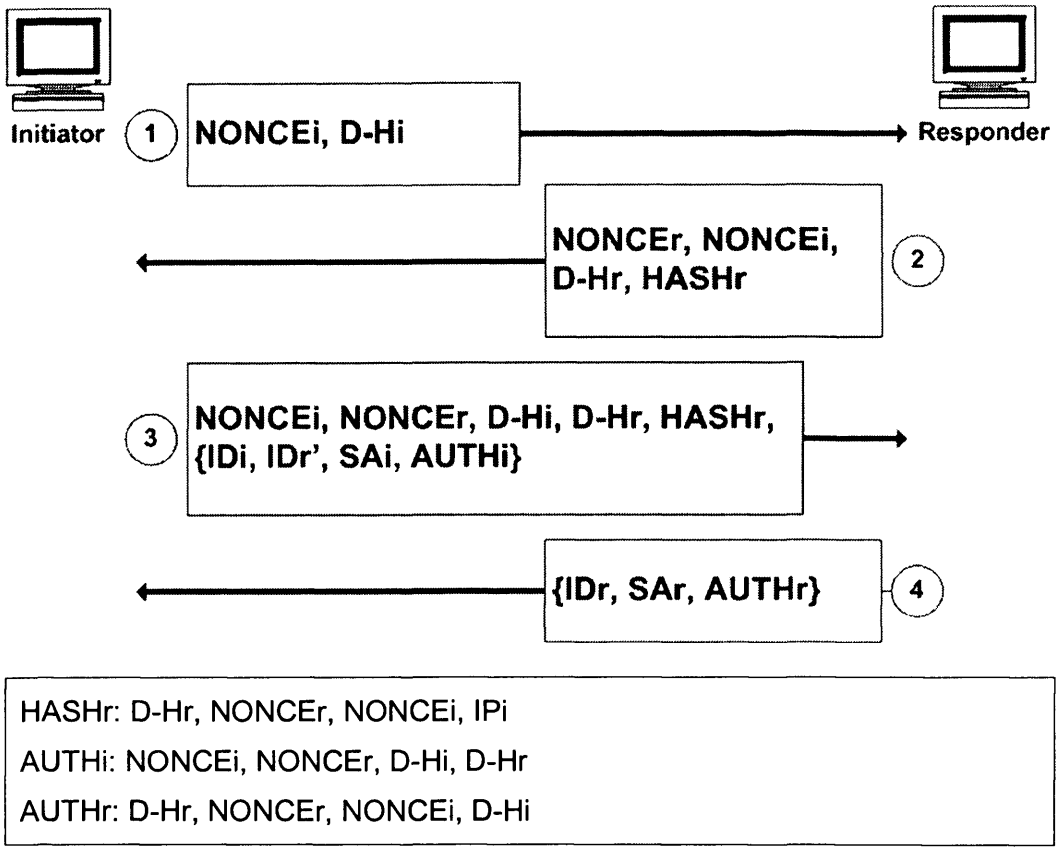

Figure 24 – The JFKr protocol

It should be noted that it is not within the interest of this thesis to justify the level of security of JFK. However, since the developers of JFK claimed JFK is DoS-resistant, JFK is studied in this thesis to enable the author to compare how the proposed protocol in this thesis and an existing solution such as JFK would tackle DoS attacks. In section 4.6 (p.110) and section 5.5 (p.150), it is discussed how JFK (and IKEv2, which is a standardised protocol) is less resistant to DoS attacks when compared to the proposed protocol in this thesis.

2.12 IKEv1 in aggressive Mode

IKEv2 is designed to be an optimised version of IKEv1 (i.e. less complex and more efficient). For completeness, IKEv1 in aggressive mode is introduced in this section. The reason to discuss IKEv1 in aggressive mode in this thesis is

because, as the readers will observe in a later section (section 5.4 on p.140), both IKEv1 in aggressive mode and the solution proposed in this thesis uses three messages to establish one hop-by-hop SA, which often gives the readers an impression that IKEv1 in aggressive mode has a similar level of scalability with the solution proposed in this thesis. This section therefore presents IKEv1 in aggressive mode, to clarify any potential mis-understandings. Note that it is out of scope of this thesis to discuss IKEv1 in more detail. Readers are referred to the cited references for more detail.

2.12.1 Two Phases Approach in IKEv1(v2)

Background information on IKEv1 can be found in [58][59]. IKEv1(v2) has two exchange phases: the first exchange phase establishes an IKEv1 SA between two nodes; the second exchange phase establishes an IPSec SA between the same pair of nodes. The established IKEv1 SA from the first phase is used to establish the IPSec SA in the second phase. The IPSec SA is used to protect packets [60].

The need for this recursive SA establishment process in IKEv1(v2) and IPSec is that IKEv1(v2) and IPSec were designed to serve different purposes. IKEv1(v2) is a key exchange protocol [61], whereas IPSec is a packet protection protocol [62][63]. An IKEv1(v2) SA is therefore used to protect key exchange processes, whereas IPSec SA is used to protect packet transmission. An established IKEv1(v2) SA between a pair of nodes can be used by multiple user applications to establish different types of IPSec SAs to accommodate different networking environments [64]. For example, if packets do not need confidentiality protection, an IPSec AH SA can be used; else, an IPSec ESP SA is used.

This flexibility provisioned in IKEv1(v2) is not needed in active networks. This is because packets must always be encrypted. Therefore, there is no need to establish multiple SAs across the same pair of hops to protect packets. As such, recursive SA establishment is not needed between active nodes. In next chapter, it will be shown that there is no need to have recursive SA establishment; in fact, a packet can be transmitted *during* a hop-by-hop SA establishment.

IKEv1 provides two options for the first exchange phase: IKEv1 in main mode and IKEv2 in aggressive mode. IKEv1 in aggressive mode is designed to be less complex, and more efficient that IKEv1 in main mode. Therefore, only IKEv1 in aggressive mode will be discussed. The features of IKEv1 in aggressive mode and IKEv1 in main mode are discussed in [65]. Note that IKEv1 provides only one option for the second exchange phase: IKEv1 quick mode.

2.12.2 An Overview on IKEv1 in Aggressive Mode

IKEv1 in aggressive mode is more efficient than IKEv1 in main mode because IKEv1 needs only half of the number of message exchanges. Only three messages are exchanged in IKEv1 in aggressive mode. An Initiator sends the first message to a Responder that includes SAi, D-Hi, NONCEi, and IDi. The responder replies with SAr, D-Hr, NONCEr, and IDr. All items in the reply are digitally signed. Finally, the Initiator sends the third and last message, which contains a digital signature that covers all items in message 1.

Readers should note that IKEv1 in aggressive mode does not protect packets: once IKEv1 in aggressive mode has been used between a pair of nodes to establish a hop-by-hop SA, the hops must go through another round of SA

establishment i.e. the IKEv1 Quick Mode Exchange. The latter exchange uses the IKEv1 SA to establish another SA for IPSec, the IPSec SA is used for protecting packets. IKEv1 Quick Mode Exchange includes an exchange of two messages, which enables the peers to negotiate and establish another SA (i.e. IPSec SA). For more detail, readers are referred to [66].

3 Security Protocol for Active Networks

It was discussed in previous sections that there is a need for a hop-by-hop security model for protecting active packets. To support hop-by-hop security, there is a need to investigate a hop-by-hop SA establishment protocol. It was also discussed in an earlier section that existing solutions suffer drawbacks in terms of efficiency, scalability, and flexibility, and more. In this chapter, a novel, efficient, scalable, and flexible solution to the problem space, known as Security Protocol for Active Networks (SPAN), is presented. Firstly, the assumptions made in the design of SPAN are discussed and justified. Secondly, the SPAN protocol is presented by beginning with an overview of the SPAN protocol, followed by the design decisions of the SPAN protocol's 3-message exchange handshake. Then, the SPAN messages are discussed individually together with their design decisions.

The SPAN protocol is designed to protect active packets with static code (section 1.6.1 on p.28); however, it is also applicable to protect dynamic code. Thus, a discussion of how the SPAN protocol is used to protect active packets with dynamic code will be presented later in this section. To ease the readers, initially, the protocol will be presented by using a simplified deployment environment (i.e. the protocol is illustrated using two nodes only); later in the chapter, the deployment of the protocol along a path (i.e. many more nodes) will be explained.

3.1 Design Assumptions

It is not within the scope of this thesis to investigate access control, or advance firewall technologies, or intrusion detection techniques, or new algorithms for

cryptography. Secure storage of keying materials and node integrity are assumed. This assumption is fundamental when designing security protocols, because no security protocol would work securely if the nodes (where the security protocol is deployed) were compromised. New execution path is assumed. It was defined that a new execution path is "*...a path which no active packets have previously traversed (hence no pre-established hop-by-hop SAs), or a path of which previously established hop-by-hop SAs has expired, therefore there is a need to establish a series of new hop-by-hop SAs along the path (hence the name new execution path)*". The assumption of new execution path in this thesis is justifiable, because currently there is no requirement to have inter-connected security channels pre-established between all nodes across the entire Internet. The applicability of SPAN in environments in which inter-connected security channels have already been pre-established will be discussed later on in section 4.8 (p.118).

This thesis assumes that active nodes have access to public key certificates, i.e. PKI, is supported. PKI is a widely deployed, standardised technology (e.g. embedded in all web browsers). Because active networks are meant to co-operate with the existing Internet, the assumption of PKI support in active networks is therefore justifiable. For simplicity, in the initial discussion of SPAN, each administrator/management application and each NodeOS is assumed to have its own PKI key pair; but provisioning have been made in SPAN to accommodate situations in which some administrator/management applications and NodeOSs do not have their own PKI key pair (section 4.7 on p.115).

Administrative issues are not addressed in this thesis (i.e. how CAs verify actual ownership of valid PKI certificates): the integrity of legitimate PKI public key pair

owners is assumed. Thus, if a person/entity uses a legitimate PKI private key for signing data, he/she would be traceable (i.e. non-repudiation protection enforced through PKI). Because node and key storage integrity and the integrity of PKI public key pair owners are assumed, it is further assumed that any requests with *valid* signatures are legitimate requests; else, they are attack messages. Attackers are assumed capable of intercepting *all* messages on the Internet, and are able to create/modify *all* types of messages that are not protected. Provisioning has been made in the proposed solution in this thesis for SA maintenance and re-keying.

3.2 An Overview on the SPAN Protocol

SPAN begins when the principal has created an active packet, and is about to inject the packet into the network. The SPAN protocol involves an exchange of three messages only to complete a hop-by-hop SA negotiation, hop-by-hop EE query, *and* a secure hop-by-hop active packet transmission (Figure 25 shows an overview of the handshake). To ease readers, the notations and terms used to describe the SPAN protocol in the following sections are adopted from the IKEv2 RFC; items that are quoted in square brackets are optional; whereas items in curly brackets are protected. All terms are explained.

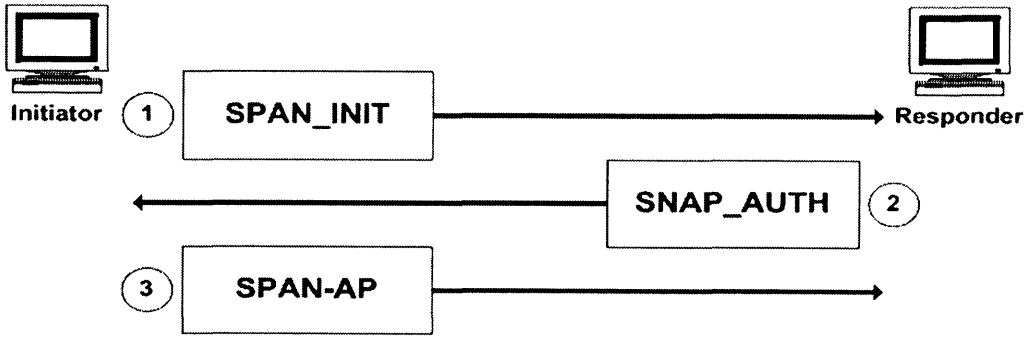

Figure 25 – The SPAN Protocol

Note that the focus of this thesis is on key management rather than designing new cryptographic algorithms. Thus, SPAN uses D-H as the algorithm to compute a shared secret; and computes a shared key set from the shared secret using the logic as described in [30].

3.3 Design Decisions for a 3-Message Handshake

The SPAN protocol has one handshake between a pair of nodes, which involves an exchange of three messages. An active packet is securely transmitted across a pair of nodes during a hop-by-hop SA establishment, instead of after. SPAN is designed with the appropriate defence mechanisms against replay attack, man-in-the-middle attack, and impersonate attack. SPAN is also designed to detect DoS attacks more efficiently than existing solutions, in order to minimise the effect of DoS attacks on victim node(s). Furthermore, SPAN is designed to improve the level of robustness and flexibility of the underlying active networking systems.

A 3-message exchange handshake is chosen for SPAN because this is the minimum number of message exchange for establishing a symmetric key between two nodes in the insecure Internet (when using the Diffie-Hellman algorithm), and to transmit an active packet. The initiator must first send a request to the Responder to indicate its incentive to establish a shared symmetric key (i.e. the 1st message). For efficiency, this message will include some data that enables the Responder to start the shared symmetric key generation process should it accept to do so. The Responder then replies to the Initiator with its willingness to establish a shared symmetric key (i.e. the 2nd message). At this stage, both nodes will have the shared symmetric key. The third message is needed to send the active packet across using the established

shared symmetric key. As a rule of thumb, active packets should be sent after a secured tunnel has been established.

3.4 SPAN Initialisation (Message 1: SPAN_INIT)

3.4.1 An Overview

Stage 1 of the SPAN protocol begins when the principal is about to inject an active packet to the network. The first message (SPAN_INIT) is sent from the Initiator to the Responder. SPAN_INIT is shown in Figure 26.

I → R:
HDR_INIT, SAi, [EEi], [CERTi], D-Hi, NONCEi, AUTHi

Figure 26 – SPAN_INIT

This first message in the exchange enables the Initiator to start a SA negotiation (SAi) and, optionally, a query on the information of a remote EE ([EEi]). Some initial key exchange values (D-Hi and NONCEi) are forwarded to the Responder; these key exchange values will be used later on for the shared secret generation. This message is digitally signed by the Initiator, and as such, this message contains a digital signature (AUTHi).

3.4.2 Design Decisions for SPAN_INIT

The purpose of the first message sent from the Initiator to the Responder is as follow:

- To tell the (potential) Responder that it wants to establish a shared key.
- To pass on necessary information to the (potential) Responder, so that should the Responder accepts to establish a shared key, it may start the shared key generation process straight away without further delay.
- To include sufficient cryptographic materials in the message so that the

Responder can verify the message authenticity and integrity.

The decisions on the design of SPAN_INIT are explained below. SPAN_INIT includes the following elements:

- SAi

This is the SA offered by the Initiator to the Responder. It contains a list of supported/preferred cryptographic algorithms of the Initiator. Thus, it must be included in the first message; otherwise, the Responder would not be able to proceed with the symmetric key generation process.

- [EEi]

This is the EE information query that the Initiator may optionally send to the Responder. This field must be included in the first message if the Initiator is uncertain about the EE on the Responder. This is because with this information, the Responder may evaluate its own characteristics, in order to decide whether to accept the Initiator's request for shared key establishment (and subsequently active packet execution).

- [CERTi]

This optional field keeps the public key certificate of the Initiator. This field should be used when the Initiator wishes to distribute its certificate to a Responder (which the Initiator is not sure whether it has its certificate). This could be used, for example, when the Initiator has recently obtained a new certificate. Multiple certificates may be placed in this field.

- D-Hi and NONCEi

These are essential pieces of information needed by the Diffie-Hellman algorithm; they are needed by the Responder to generate the shared key. Including this information in the first message enables the Responder to start

the key generation process after receiving the initial request from the Initiator.

- AUTHi

This is the signature created at the source node. This signature must be included in the first message so that the Responder can verify the authenticity, integrity, and non-repudiation of the message from the Initiator prior to carrying out any further computation. This is an important step in defending the system from DoS attacks (see section 4.6 on p.110).

3.4.3 SPAN_INIT in Detail

More specifically, HDR_INIT is the SPAN_INIT message header. SAi is a set of security association parameters offered by the Initiator to the Responder. These parameters are for example the supported or preferred encryption algorithms (of the Initiator), and the supported or preferred key size (of the Initiator). D-Hi and NONCEi are the Diffie-Hellman public value and a random 128-bit, never reused nonce generated by the Initiator. Both D-Hi and NONCEi are required for the symmetric secret establishment between the Initiator and the Responder (section 2.10 on p.55). Also, the nonce will be needed for anti-replay attacks (section 4.2 on p.100).

[CERTi] is the PKI certificate of the Initiator. Distributing the Initiator's PKI certificate during the SPAN SA establishment process is optional, and depends on the choice of the Initiator. This is because the Responder may already have obtained the Initiator's PKI certificate through previous SA establishments (but now the previously established SAs have expired), or by other means e.g. pre-distributed PKI certificates using out-of-band channels. As discussed in the assumption, administrative issues such as PKI certificate distribution and maintenance are out-of-scope of this thesis. As such, the SPAN protocol does

not explicitly enforce PKI certificate distribution during SPAN SA establishment; but provisioning has been made in the protocol to accommodate situations in which PKI certificate (re)distribution is needed. This field, may be used for distributing more than one certificates. For example, a node may distribute certificates of other nodes, if the (protected) content of the packet that it is about to send to other nodes would need those certificates for verification.

[EEi] is needed to enhance the level of robustness of the underlying active networking systems. It is optional, and it is included only when the Initiator needs to confirm that the Responder does satisfy certain requirements that are needed to execute the to-be-sent active packet, *prior to* establishing a hop-by-hop SA and sending over the active packet. An example query would be the availability of specific supportive software/service modules (that are required to execute the active packet), or programming language supported by the remote execution platform. It will be discussed in a later section (section 4.5 on p.106) that the use of [EEi] improves the level of robustness of the underlying active networking systems.

The Initiator must provide authenticity, integrity, and non-repudiation protection for SPAN_INIT by digitally signing this message using its PKI private key. AUTHi is a digital signature that is created by using the Initiator's private key that covers all the items contained in SPAN_INIT except CERTi[8] and the digital signature itself. Figure 27 shows the items digitally signed by the Initiator.

HDR_INIT, SAi, [EEi], D-Hi, NONCEi

Figure 27 – Items digitally signed in AUTHi

[8] Note that [CERTi] is not signed explicitly in AUTHi because certificates should be verified through PKI.

The inclusion of AUTHi in SPAN_INIT enables more efficient detection of DoS attacks (section 4.6 on p.110). Note that because [EEi] is digitally signed, SPAN enables the principal to make *authenticated* and *integrity protected* queries for remote EE information. Shortly, it will be discussed how SPAN enables the Initiator to receive protected replies to [EEi], prior to active packet transmission (section 3.5 on p.82).

3.5 SPAN Authentication (Message 2: SPAN_AUTH)

3.5.1 An Overview

Stage 2 is carried out at the Responder. During this stage, the Responder verifies SPAN_INIT (message 1 from the Initiator), and computes a shared secret, and subsequently computes a shared key set based on its own data and the data provided by the Initiator. The Responder then responds to the Initiator with the data that are needed by the Initiator to complete the shared secret computation process at the Initiator's side. The reply message from the Responder also contains some replies of the Responder to the EE queries that were made by the Initiator in SPAN_INIT. Parts of SPAN_AUTH are digitally signed by the Responder; other parts are protected by the freshly created shared key set for security reasons (section 4.3 on p.103).

I ← R:
HDR_AUTH, SAr, [CERTr], D-Hr, NONCEr, AUTHr, {[EEr], IDr}

Figure 28 – SPAN_AUTH

3.5.2 Design Decisions for SPAN_AUTH

SPAN_AUTH is designed to enable the Responder, using just one message, to notify the Initiator that:

- It has accepted the Initiator's request to generate a shared symmetric key.

- It has generated the shared symmetric key.
- It wants to enable the Initiator to generate the shared key.
- It wants the Initiator to validate the shared key.

The decisions for including the corresponding elements in SPAN_AUTH are as follow:

- SAr

This must be included in this response so that the Initiator knows which of its supported/preferred cryptographic algorithm(s) the Responder has selected to use. Without this information, the Initiator would be unable to proceed with the shared key generation process.

- [CERTr]

The Responder may optionally sends its public key certificate to the Initiator if it is unsure whether the Initiator has its public key certificate from (if there was any) previous interactions. Note that SPAN_AUTH is the only message sent from the Responder to the Initiator; thus, this is the only chance that the Responder could send anything it wants to send to the Initiator.

- D-Hr & NONCEr

These are the key elements needed by the Diffie-Hellman algorithm in order to generate the shared key. These elements have been used by the Responder to generate the shared key; thus, the Responder must send this information to the Initiator, so that the Initiator may generate the shared key.

- AUTHr

This is the signature created by the Responder. It protects the message and therefore must be included. Furthermore, it acts as a countersign of some elements in SPAN_INIT to tackle network attacks such as impersonation

attacks (section 4.2 on p.100).

- {[EEr], IDr}

This is a protected payload, which includes information on the Responder's EE, and some identification information that the Responder wants to use in the future. The Responder's EE information should be included if the Initiator has made an EE information request in SPAN_INIT, or the Responder wants to tell the Initiator some information of its EE. This information is included to enhance the efficiency, robustness, and security of the SPAN protocol (see next section for detail). They are included in this message because this is the only message sent from the Responder to the Initiator.

Note that these items are protected by using the recently generated shared key. The Initiator must verify these items (after it has generated the shared key). By verifying these items, the Initiator is able to verify that the Responder has computed the symmetric key correctly (see section 4.3 on p.103).

3.5.3 SPAN_AUTH in Detail

More specifically, upon receiving the first message (i.e. SPAN_INIT) from the Initiator, the Responder verifies the authenticity, integrity, and non-repudiation of the digitally signed materials in SPAN_INIT by using [CERTi] and AUTHi. If the digitally signed items cannot be verified, the Responder stops proceeding further because SPAN_INIT might have been subjected to man-in-the-middle attacks, or was created for DoS attacks (section 4.6 on p.110).

If the signature is verified (hence the contents of SPAN_INIT), the Responder will look into the message. If a [EEi] is included, the Responder will evaluate itself against the list of requirements carried in [EEi]. There are two possible outcomes:

1. The Responder is *unable* to satisfy to the requirements as specified in [EEi]. This could happen for example when the Responder has already been re-configured such that it cannot execute the to-be-sent active packet; or the Responder has an incompatible execution platform for the packet, or the Responder does not have the necessary supportive software modules or services to execute the packet. In this case, the SPAN_INIT packet is simply forwarded to the Responder's neighbouring node (a node other than the one from which the active packet has arrived from, otherwise the packet will be travelling in a loop), where the SPAN protocol exchange may potentially continue.
2. The Responder is able to respond to the requirements as specified in [EEi]. For example, it satisfies all the requirements specified by the Initiator, or it has some missing supportive software modules or services, but it believes these missing supportive items can be provided by the Initiator. The Responder creates a list of its replies, and stores them in an [EEr] payload. This facility is important in SPAN to enhance robustness, it enables the Initiator to pass on additional information to the Responder (in the last message of the exchange, see later) to ensure that a smooth execution of active packets (see section 4.5 on p.106).

In case 2 or in the case where no [EEi] is included in SPAN_INIT, the Responder generates its own D-H public value (D-Hr) and a random 128-bit nonce (NONCEr). By using these values in conjunction with the Initiator's values, i.e. D-Hi and NONCEi, the Responder is capable of creating a shared secret (SKEYSEED) using the D-H algorithm (section 8.6 on p.168). Note that SKEYSEED is a shared secret established secretly between the Initiator and

the Responder, from which a subsequent shared key set can be generated for specific purposes. For example, the authentication key (SK_a) is one of the keys in the subsequently computed shared key set which is used for authenticity protection and integrity checks; the encryption key (SK_e) is used for encryption.

Once the Responder has computed the shared secret and the shared key set, it responds to the Initiator with the second message in the SPAN protocol (SPAN_AUTH), which is shown in Figure 28. Note that the items quoted in curly brackets {...} are protected accordingly as embedded payloads in the same Encrypted payload, by using the corresponding shared key derived from the shared secret SKEYSEED i.e. SK_e. The Encrypted payload is appended with integrity protection data – in this case a keyed hash value – that covers SPAN_AUTH (including the message header and payload). The key used to create the keyed hash value (i.e. the integrity protection data) is the SK_a key. HDR_AUTH is the message header of SPAN_AUTH. SAr keeps the Responder's replies to SAi (e.g. the Responder's choice of encryption algorithms offered by the Initiator). [CERTr] is the public key certificate of the Responder, again for the same reason as explained above, the inclusion of [CERTr] is optional and depends on the Responder's own choice. D-Hr and NONCEr are needed by the Initiator to create the shared secret SKEYSEED, and therefore must be listed in cleartext i.e. not encrypted. AUTHr is a digital signature created by using the Responder's private key over a list of items.

HDR_AUTH, SAr, D-Hr, NONCEr, NONCEi

Figure 29 – Items digitally signed in AUTHr

The idea of digitally signing the items listed in Figure 29 by the Responder, is to

enable the Initiator to verify the authenticity, integrity, and non-repudiation of the shared secret establishment parameters from the Responder. Note that the Initiator's nonce (NONCEi) is also digitally signed by the Responder. This arrangement is necessary to prevent replay attacks (section 4.2 on p.100). Also note that the Responder does not simply sign any anonymous nonce values. The Responder must first verify NONCEi (that was included in SPAN_INIT), by verifying the value against the digital signature (AUTHi) created by the Initiator, *prior to* digitally signing NONCEi.

{[EEr], IDr} is the protected replies to [EEi] and the identity of the Responder that will be used for future identification respectively. They are protected by using the appropriate shared key i.e. SK_e and SK_a. IDr is not necessarily the identity of the Responder as listed in [CERTr]. It could be any form of identifier (e.g. IP addresses and host names) that the Responder considers appropriate to be used in future for identifying itself. This identity information may also be an identity of an AA/EE, which is injecting active packets to the network, but relying on the active node to provide hop-by-hop security for its active packets. Note that this information is protected for two reasons:

1. The EE replies and the Responder's identity may contain sensitive information (such as real-time operational status of the EE on the Responder), so they should be protected from intruders.
2. The protected payload can be used by the Responder as a proof-of-knowledge of the shared key set (section 4.3 on p.103).

3.6 SPAN Active Packet (Message 3: SPAN_AP)

3.6.1 An Overview

Stage 3 is carried out at the Initiator. During this stage, the Initiator first verifies

the digital signature of the reply message from the Responder (i.e. AUTHr in SPAN_AUTH), then computes the shared secret (SKEYSEED), and subsequently computes the shared key set by using its own data and the data provided by the Responder in SPAN_AUTH. The shared key set is then used to verify other parts of SPAN_AUTH (i.e. the items in the curly bracket). Then, the Responder's reply (i.e. [EEr]) to the Initiator's initial EE queries (i.e. [EEi]) is extracted.

To complete the protocol, the Initiator sends to the Responder the third (protected) message (SPAN_AP) which contains the active packet and, optionally, some additional data that would enable a successful execution of the active packet at the Responder: for example, the missing modules that are requested by the Responder in [EEr]. These modules are for example an additional Java class (needed by the Responder) to support the execution of the Java-based active code. The modules will be included into the reply if they can be fit into the reply message; else, a link that refers to a location where the modules can be downloaded from will be included instead. This is an example of an out-of-band approach in active networks [14]. The SPAN_AP message is shown in Figure 30.

I → R:
HDR_AP, {IDi, NONCEr, ACTIVE_PACKET, CODE_SIG}

Figure 30 – SPAN_AP

3.6.2 Design Decisions for SPAN_AP

This is the last message in the protocol. The purpose of this message is to enable the Initiator to tell the Responder:

- It has generated the shared symmetric key.

- It wants the Responder to validate the shared key.
- It is passing the (protected) active packet to the Responder using the recently generated shared key and asymmetric cryptography.

The decisions for including the corresponding elements in SPAN_AP are as follow:

- {[IDi]}

Some (protected) identity information of the Initiator that the Initiator wants the Responder to use for future communications. It is confidential, thus the Initiator must only send this information after it has verified the Responder's authenticity (i.e. after checking on the Responder's message). Thus, this information must be sent in this message (not in message 1 because at that point the Initiator had not verified the Responder's authenticity).

- {NONCEr}

This is a countersigning process. The Initiator must countersign some elements of the Responder's message to prevent network attacks such as impersonation attacks (section 4.2 on p.100).

- {ACTIVE_PACKET} & {CODE_SIG}

The protected active packet and the digital signature on its static code. They are sent at this stage because only at this stage the Initiator has verified the Responder. The code signature is needed because it is for enforcing non-repudiation protection on the code. Thus, the code's original creator, i.e. the principal, cannot deny of any wrongdoing should the code cause any damage to other nodes.

Note that these items are protected (i.e. encrypted and integrity protected) by using the recently generated shared key. They must be protected because they

may contain node operation sensitive information. The Responder must verify these items. By verifying these items, the Responder is able to verify the Initiator has computed the symmetric key correctly (see section 4.3 on p.103).

3.6.3 SPAN_AP in Detail

In more detail, upon receiving SPAN_AUTH from the Responder, the Initiator must first verify AUTHr using [CERTr]. If the verification process were successful, the Initiator would be able to generate the shared secret, i.e. SKEYSEED, and the subsequent shared keys (i.e. SK_e, SK_a) by using D-Hi, D-Hr, NONCEi, and NONCEr. The Initiator then starts to verify the authenticity and integrity of the items contained in SPAN_AUTH: the Initiator decrypts the encrypted items in SPAN_AUTH i.e. {[EEr], IDr}, by using the corresponding shared key i.e. SK_a and SK_e respectively. If SPAN_AUTH is verified, the Initiator sends to the Responder the third, and the last message: SPAN_AP.

HDR_AP is the header of SPAN_AP. IDi is the identity of the Initiator (or the AA/EE that it is representing). NONCEr is protected so that the Initiator can acknowledge to the Responder that it has received the correct nonce value and for anti-replay attacks (section 4.2 on p.100). This value does not need to be digitally signed by the Initiator, because it is protected by the authenticated and integrity verified shared keys i.e. SK_e and SK_a. ACTIVE_PACKET contains the *entire* active packet i.e. both the static code and dynamic data. The dynamic data is the result of static code execution on the Initiator. CODE_SIG is the signature on the static code that is created for protecting the authenticity, integrity, and non-repudiation of the static code, by using the *principal's* private key. Note that the principal is the actual creator of the code. In situations where the principal does not have its own public key pairs, the private key of the node

that the principal is currently residing may be used for signing instead. This arrangement is more scalable at the expense of a less than ideal non-repudiation protection (section 4.7 on p.115). Note further that, by using the optional [CERTi] field, a node may pass onto other nodes the public key certificate of itself, as well as the certificate(s) that the node has obtained from other nodes in previous communications.

Once the Responder has received SPAN_AP, the protected items in the message are subjected to verification by using the established shared key set. Firstly, the integrity data appended to SPAN_AP is verified. Then, the encrypted payload in SPAN_AP is decrypted. The authenticity and integrity of the static code in the active packet are verified by checking the digital signature (CODE_SIG) in SPAN_AP, which was created by the principal using the principal's private key. If the verifications are successful, the embedded code in the active packet is executed.

3.6.4 Secured Active Packet Transmission

Under this arrangement, the hop-by-hop authenticity, integrity, and confidentiality of the dynamic data of the active packet are protected by the shared key set. The static code is digitally signed by the principal, so the source authenticity, integrity, and non-repudiation of the static code are protected. The confidentiality of the entire active packet including both static code and dynamic data is protected by the shared key set.

3.7 Multiple Hops Transmission

In previous sections, the SPAN protocol was presented using a simplified deployment environment with two nodes only (i.e. an exchange between an Initiator and a Responder). In reality, the Responder (which is an intermediate

node) would want to pass on the received packet to other nodes in the network (i.e. a multiple hops transmission). Figure 31 shows the differences between the two.

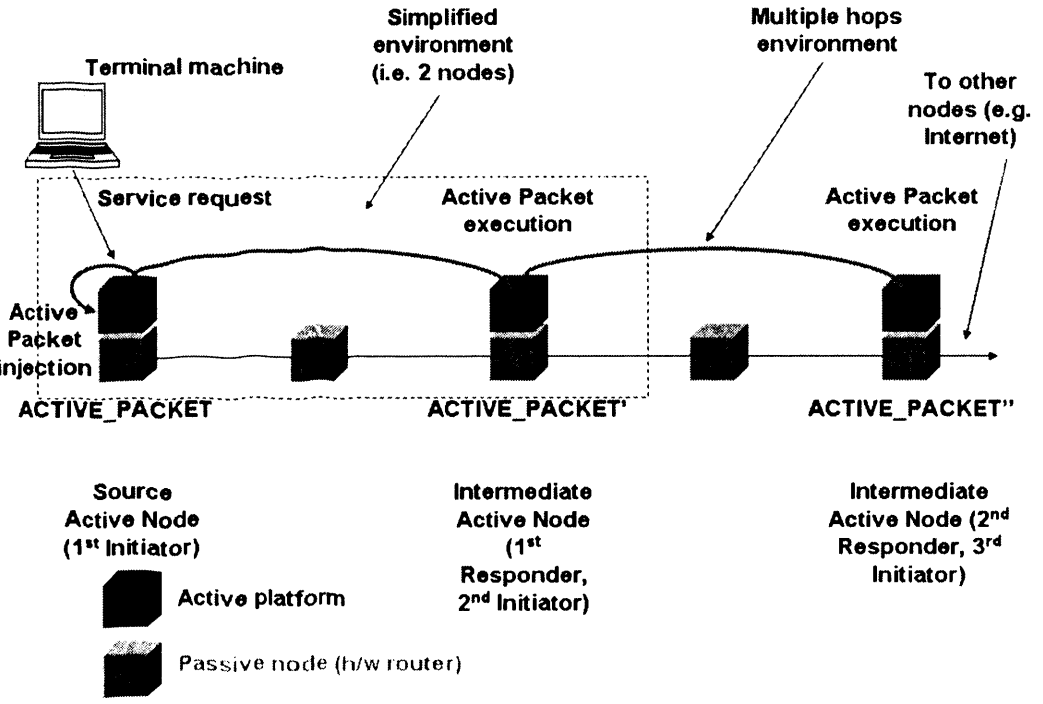

Figure 31 – Multiple hops transmission

The same SPAN protocol is used for multiple hops transmission. Once the first Responder (i.e. the node that first intercepted the active packet from the source node) has executed the active packet, the results of code execution, i.e. new dynamic data, will be added back to the packet, and the packet will be forwarded to its next hop i.e. the 2nd Responder. More specifically, when sending the active packet to another node, the first Responder now becomes an Initiator, i.e. the 2nd Initiator (Figure 31). This is because it is now attempting to start a secure transmission with another node.

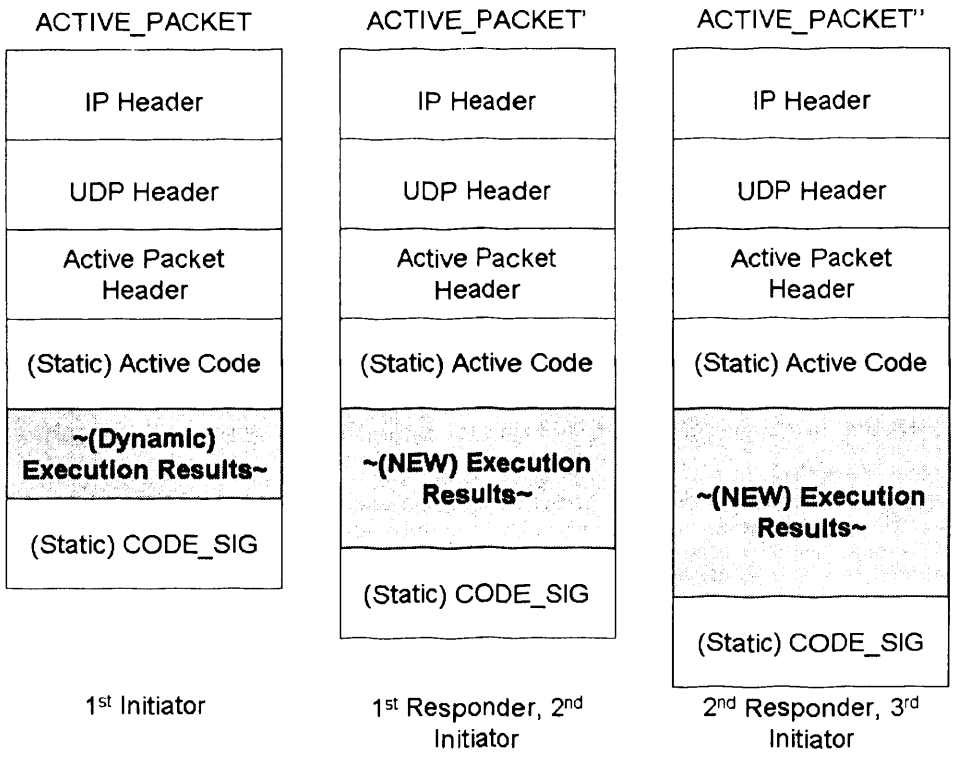

Figure 32 – Active packet at each node

The SPAN protocol repeats between the new pair of nodes. Figure 32 shows the active packet at each node. The new active packet (i.e. ACTIVE_PACKET') from the second Initiator would contain the *same* static code (i.e. STATIC_CODE) as the packet received from the first Initiator, but with *new* dynamic data, which contains the execution results. Note that the 2nd Initiator should *not* generate a new signature on the static code; this is because the code is static (i.e. to be executed on all nodes) and should be verified by verifying the *principal's* signature on the static code i.e. based on the identity of administrator/management application on the source node. If there was a need to modify the static code (that was originally created by the principal), the second Initiator would have to create a *new* active packet (see section 3.9 on p.94). The same process repeats between the third Initiator and other nodes

until the packet reaches its destination.

3.8 Protecting Active Packets in Subsequent Communications

Once a SA has been established between a pair of nodes, the SA should be retained until a *timeout* (section 8.16 on p.189). Future active packet transmissions, remote EE queries and replies, i.e. [EEi] and [EEr], between the pair of nodes may make use of the established hop-by-hop SA. The established SA is identified at each node by the respective Security Parameter Index (SPI) (that was assigned by the nodes involved during the SA establishment process) (section 8.17.1 on p.192). The SPI is also used for identifying established SAs during (future) re-keying. Thus, subsequent packets are protected by using an SPAN header (which is similar to IKEv2 header), which contains the SPIs for identifying the established SAs to be used, and a 32-bit cryptographically protected message ID (for anti-replay attacks). The entire packet is covered by an integrity checksum data.

3.9 Protecting Active Packets with Dynamic Code

In section 1.6.1 (p.28), it was discussed that in some occasions, an active node may want to modify the original code of an intercepted active packet (i.e. dynamic code). It was also emphasised that the SPAN protocol may be used to protect active packets with static code or dynamic code. As a recap, static code refers to the code that remains unchanged during the packet's traversal in the networks; whereas dynamic code means the code is changed during the packet's traversal. In this section, the use of the SPAN protocol for protecting active packets with dynamic code is discussed.

In Figure 6, the content of an active packet traversing the network is shown. Note that at all nodes, the active code in the packet remains unchanged. In

Figure 33, the content of an active packet (without protection) with dynamic code is shown. Note that at each node, the active code is modified (the grey boxes).

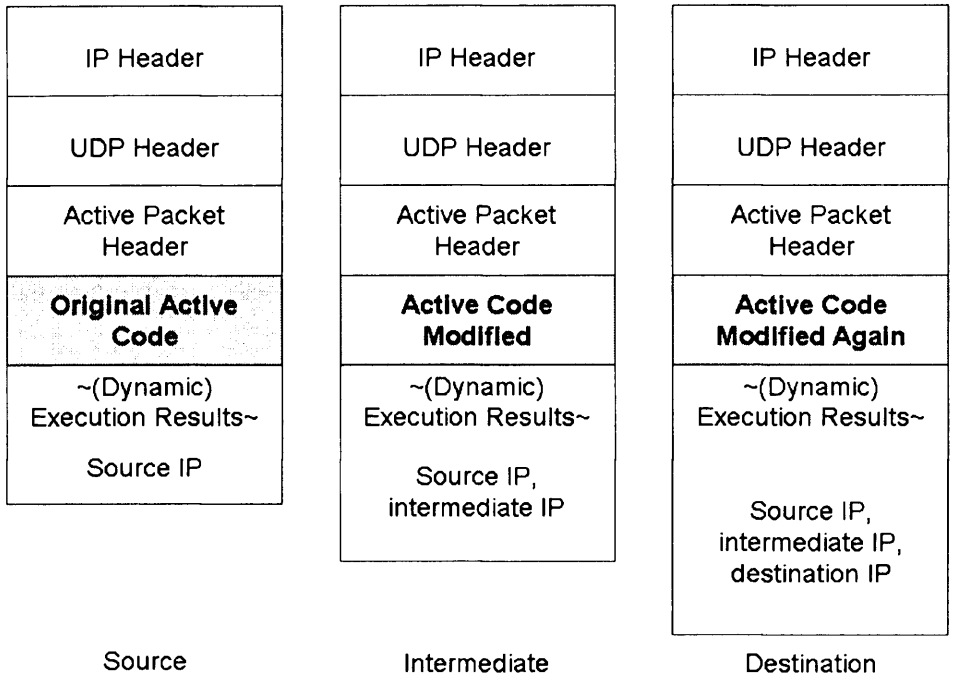

Figure 33 – Dynamic code in active packets using the same example in section 1.5.3 (p.24)

It was discussed in section 3.6.2 (p.88) that active code must be digitally signed by its original creator, i.e. the principal, to enforce non-repudiation. As such, if dynamic code is created at an intermediate node, the original creator of this dynamic code (e.g. an EE or a management application currently residing the intermediate node) must digitally sign its code. This means that, when a node generates dynamic code, the node is essentially creating a *new* active packet. The node will simply use the SPAN protocol (in the same way as described earlier in this chapter) to transmit this new active packet to its next hop.

3.10 Packet Loss Handling in SPAN

Packet loss in SPAN can be handled through retransmission using standard techniques (see shortly later). Designing specific packet loss handling mechanisms in SPAN is not within the scope of this thesis. The reason is that there are existing packet loss handling methods that are being widely used in today's transmission models. For example, TCP implements reliable retransmission. Note that SPAN and IKEv2 are protocols on the application layer. They are on the application layer because they are protocols designed to enable security managers (i.e. users) to modify the behaviour of lower levels in the stack through, say, changing encryption policies such as IPSec SAs (i.e. layer 3). Thus, SPAN can be run over UDP or TCP. TCP is a connection-oriented protocol, which supports retransmission. On the other hand, UDP is connectionless, but with less overhead. If SPAN is run over UDP, a retransmission technique similar to the one used in IKEv2 can be used in SPAN to handle packet loss: the Initiator and the Responder remember their messages, and retransmit after a timeout. They may discard the messages after the exchange has completed.

3.11 Summary

In this chapter, the SPAN protocol was presented together with its design decisions. SPAN is a fully distributed protocol that allows each participating active node in the network to establish a hop-by-hop SA with its neighbouring active node. The protocol involves a three-message exchange handshake; each exchange is specially designed to enhance scalability, efficiency, and flexibility of the protocol. SPAN also enhances the robustness of the underlying active networking systems, by ignoring incompatible active nodes. Furthermore,

an active packet is transmitted during the SPAN's hop-by-hop SA establishment process, instead of after. In chapter 4 (p.98), a discussion of SPAN and a comparison between SPAN with existing solutions will be presented.

4 Discussion

In this section, related work and certain unique features of SPAN that make SPAN ideal for hop-by-hop security management in active networking systems will be discussed.

4.1 Message Security in SPAN

All messages exchanged in SPAN are secured. They are secured using different techniques. In this section, these techniques will be discussed in detail.

4.1.1 Message Authenticity, Integrity and Confidentiality Protection

- Secured message content via Symmetric Cryptography

The authenticity and integrity of the keying materials from the Initiator to the Responder (i.e. items in SPAN_INIT) are verifiable to the Responder because the Initiator signs the message contents (except its PKI certificate, which should be verified through PKI) with its own private key. Similarly, the authenticity and integrity of the Responder's keying materials are verifiable to the Initiator because the Responder signs the contents of SPAN_AUTH with its own private key.

- Anti-network attack via asymmetric cryptography

The Responder signs the Initiator's nonce, i.e. NONCEi, to prevent replay attacks (section 4.2 on p.100). The Initiator also signs the Responder's nonce, but it does not use asymmetric cryptography. There is no need for the Initiator to digitally sign NONCEr; this is because NONCEr is authenticated and integrity protected by using the shared key set, i.e. SK_a, that was derived from *authenticated* an *integrity verified* keying materials (note that the keying materials are digitally signed/counter-signed by each peer). Note that these

keys are derived from the shared secret that was computed using D-H; and as such, they are known to the Initiator and the Responder.

- Integrity and confidentiality protection to message content

Note that the SPAN_AUTH message and the SPAN_AP message contain an Encrypted payload respectively i.e. {[EEr], IDr} and {IDi, NONCEr, ACTIVE_PACKET, CODE_SIG}. The Encrypted payload is appended by a keyed checksum, which covers the *entire* SPAN_AUTH and SPAN_AP message respectively. The keys used to create the Encrypted payload and the checksum are derived from the shared key set (SKEYSEED) i.e. SK_e and SK_a. Thus, the hop-by-hop authenticity, integrity, and confidentiality of all items in {...} are protected.

- Protected compatibility queries

As explained in an earlier section, optionally, the Initiator may wish to explore the properties of the Responder (by using the [EEi] option), *prior to* establishing a hop-by-hop SA and sending over an active packet. As such, the SPAN protocol provisions for EE information query in its first message SPAN_INIT i.e. [EEi]. The content of this enquiry is not encrypted but authenticated and integrity protected because it is digitally signed by using the Initiator's private key. For flexibility, SPAN does not specify the type of remote EE information that is allowed to be queried through this request message. The supported type of request is dependent on the static code execution requirements, and should be defined by the corresponding administrator/management application. Generally, the query should contain non-sensitive information. If sensitive information must be included in the Initiator's EE information query message, [EEi] must not be included in SPAN_INIT. In fact, such query should only be

made *after* the shared secret has been established, or by other secured means (e.g. encrypted with the receiver's public key).

In SPAN_AUTH, SAr, D-Hr, and NONCEr are carried in cleartext. This arrangement is necessary because these values are needed by the Initiator in order to generate SKEYSEED (using D-H). According to the D-H algorithm, the disclosure of these values is necessary and does not create risk to the key exchange process. For confidentiality protection on the replies on EE information query and the Responder's identity – which could be sensitive – they are required to be placed in an Encrypted payload. In SPAN_AP, the confidentiality of the Initiator's ID, and the entire active packet (and the static code signature) is protected by using the corresponding keys from the shared key set.

4.1.2 Summary

- All messages are secured in SPAN.
- A mixture of techniques is used to achieve different levels of security in SPAN (i.e. asymmetric cryptography is used to achieve non-repudiation protection on static code, symmetric cryptography is used to achieve scalable integrity, authenticity, and confidentially protection).
- The SPAN protocol enables secured compatibility enquires prior to actually establishing secure tunnels between nodes.

4.2 Network Attacks on SPAN

When designing key exchange protocols, the design of an anti-network attack mechanism is crucial. Example network attacks are replay attacks, impersonate attacks, and man-in-the-middle attacks. In this section, the anti-network attack techniques used in SPAN will be discussed.

4.2.1 Anti-Network Attack Techniques in SPAN

- Replay attacks

A typical form of replay attack is that the attacker copies a legitimate message, and re-sends the message to one of the peers or other peers. To provide anti-replay protection in SPAN, all messages exchanged in SPAN are cryptographically protected. Particularly, randomly generated, never reused, authenticated and integrity protected 128-bit nonces are included.

The use of nonces in key exchange is a generic technique to counter replay attacks (by adding randomness to the protocol). The basic idea is that the nonce is randomly generated and will not be re-used. As such, if a receiver (e.g. a Responder) receives two (or more) messages using the same nonce, the more recently received message(s) will be considered as replay attack(s), and will be dropped. Because the nonce is cryptographically protected, attackers cannot modify the nonce of a valid message unless he/she has the legitimate key (note that key and node integrity are assumed in this thesis).

- Impersonation attacks

Impersonation attacks [67] deceive the identity of legitimate nodes. This type of attack takes place in key exchange when nonces are not countersigned by the peers. For example, an Initiator sends a signed request (that contains, say some D-H public values and a nonce) to a Responder. An attacker intercepts this request, and responds with a legitimate reply. However, the reply could be a previous legitimate reply message from the Responder to *another* Initiator that was (previously) intercepted by the attacker. Because nonces are not countersigned, the Initiator would believe that the attacker is the legitimate

Responder[9].

To avoid an impersonation attack, each peer in SPAN must either digitally sign or use symmetric cryptography to protect the authenticity and integrity of each other's nonce: the Responder digitally signs the Initiator's nonce in SPAN_AUTH, and the Initiator protects the authenticity and integrity of the Responder's nonce in SPAN_AP by using the SK_a key. In this case, impersonation attacks would fail, because a legitimate reply message from the Responder must include the Initiator's nonce, unless the attackers have the PKI key pair of the legitimate Responder.

- Man-in-the-Middle attacks

Man-in-the-Middle attack is not possible in SPAN, as long as private keys are kept securely on the nodes (which is assumed). This is because PKI certificates are used for verifying AUTHr and AUTHi at the Initiator and the Responder respectively. When a forged SPAN_AUTH is received by the Initiator (i.e. SPAN_AUTH with a forged signature), the Initiator would be able to determine immediately that the message is forged because the forged signature (i.e. the forged AUTHr) cannot be verified against the certificate of the legitimate Responder (i.e. the actual owner of [CERTr]). The rule-of-thumb deployed in SPAN is that the communicating peers must be verified to each other (through verifying AUTHi and AUTHr) *prior to* active packet transmission.

4.2.2 Summary

- SPAN addresses replay attacks by including randomly generated, never re-used nonces in exchanged messages.

[9] The consequence of an impersonation attack does not necessary lead to an attack on the key exchange itself (because the use of D-H would prevent a third person from determining the shared secret). However, the attacker would have successfully "impersonated" the Responder i.e. the Initiator is tricked to believe the attacker is the Responder, which is a form of identity thieving.

- Impersonation attacks against SPAN are not possible because SPAN requires communicating peers to countersign each other's nonce.
- Man-in-the-middle attacks are not possible in SPAN, as long as keys are kept securely on nodes.

4.3 Proof-of-Knowledge of Shared Keys

In the second message of SPAN (i.e. SPAN_AUTH), IDr and [EEr] are protected by using the established shared key set. Besides confidentiality protection on nodes' identity and EE information, this is a precaution step to ensure that correct computation of the shared key set at both peers.

It should be noted that the Initiator and the Responder are trying to compute a shared secret over an insecure channel without any pre-knowledge. Therefore, the SPAN protocol requires the peers to use the freshly created shared key set *as soon as* the keys are computed. This arrangement enables the peers to verify to each other that they own the same pair of shared keys (i.e. correct computation) as soon as the shared key sets are computed, prior to using the shared key set for subsequent communications. Thus, peers can detect - and possibly correct - frauds of the protocol exchange at the earliest stage.

In SPAN, the Responder is required to use its shared key set to protect some data as soon as it has computed the shared key set; and the Initiator is also required to use its shared key set to verify the protected data (from the Responder) as soon as it has computed the shared key set. More specifically, when the Initiator receives SPAN_AUTH from the Responder, it computes its shared key set. Then, it must verify the protected contents in SPAN_AUTH by using its shared key set. Because these contents are protected by the Responder using the Responder's shared key set (which should be the same

as the Initiator's shared key set if the computation is correct), the Initiator would know the Responder has computed the same shared key set as it had. The Responder gets the same assurance by verifying the protected contents in message 3 (SPAN_AP) from the Initiator.

4.3.1 Summary

- Communicating peers are required to validate each other's key as soon as their keys are computed. This requirement enables peers to discover (if any) key computational error at the earliest stage.

4.4 Identity Protection

Some level of information disclosure during key exchange is not avoidable [68]. When computing shared secret key, SPAN follows the same guidance as recommended in [69]. The general rule is not to disclose *sensitive* information.

In IKEv2 and IKEv1 in aggressive mode, the identity information of the Initiator would be disclosed to the Responder before the Initiator can verify the Responder's authenticity (section 2.10 on p.55 and section 2.12 on p.70 respectively).

- Identity disclosure in JFK

In JFK, two variants of the protocol were provided i.e. JFKi and JFKr (section 2.11 on p.67). However, neither of the protocols protects the identity of *both* peers. JFKi protects the Initiator's identity only. The Responder's identity is disclosed to the Initiator in JFKi message 2 prior to the Responder is able to verify the Initiator's authenticity (that can only be achieved after receiving JFKi message 3). In contrast, JFKr protects the Responder's identity only. The Initiator's identity is disclosed to the Responder in JFKr message 3 prior to the Initiator being able to verify the authenticity of the Responder (that can only be

achieved after receiving JFKr message 4). In SPAN, however, the nodes may disclose the (protected) identities only *after* verifying the other node's authenticity i.e. after checking the AUTHs. Thus, identity protection is enforced in SPAN.

- The reasons for protecting identities

One may argue that the node's identity would have been disclosed if the node were required to create a digital signature. Note that the identity information that is protected is not necessary the node's identity. The Initiator is located on the NodeOS of an active node, and (together with other on-node security facilities) is responsible for protecting active packets on behalf of EEs or AAs. Thus, the protected identity information could be provided by respective EEs or AAs, which requested the Initiator to secure their active packet. This information, should be protected (so that the EE/AA remains anonymous), until the communicating Initiator and Responder can verify each other's authenticity.

- The techniques used in SPAN for identity protection

Note that in SPAN, no identity information is released through message 1 (SPAN_INIT) from the Initiator to the Responder. This is because the Initiator has not verified the Responder's authenticity. The EE/AA's identity (i.e. [IDr]) is protected in message 2 (SPAN_AUTH), that is sent by the Responder to the Initiator. This is because by verifying message 1, the Responder would have verified the authenticity of the Initiator. The Initiator only release the identity information (i.e. [IDi]) to the Responder in message 3 (i.e. SPAN_AP), after the Initiator has verified message 2 (i.e. SPAN_AUTH) from the Responder.

4.4.1 Summary

- Identities must be protected in SPAN for ensuring the actual

communicating applications on remote nodes may communicate in an anonymous fashion (when necessary).

- SPAN enables communicating applications to communicate in an anonymous fashion by requiring nodes to disclose identity information using protected message payload.

4.5 Enhanced Robustness, Flexibility, and Scalability

4.5.1 Enhancing Robustness in SPAN

The [EEi] and [EEr] options are needed for enhancing the robustness of the underlying active network systems. For example, if a principal or an Initiator is about to send an active packet for aggregating network-wide information regarding the top ten heaviest flows in the network, the principal/Initiator may need certain information regarding a specific EE on the remote node (i.e. a QoS-monitoring EE on the Responder) prior to establishing a hop-by-hop SA with that remote node, and sending over the packet. Information such as implementation information regarding the EE on the Responder, or the programming language(s) supported by the Responder, or availability of special software modules that are required to execute the active packet. The [EEi] and [EEr] options provide an opportunity for a principal/Initiator to explore their neighbours' properties prior to passing over active packets. To illustrate further, suppose Java is used (i.e. the active platform is written in Java, the EE is implemented in Java, so as the active code is in Java). To execute some active code (e.g. HelloWorld.class), a supportive Java class is needed (e.g. the SayHello.class that contains the sayHelloWorld() method). Assume that the Initiator has previously launched the same code (so it assumes the code is somewhere in the network), and decides not to include the supportive class with

the active code in order to save packet space. The Initiator can make use of the [EEi] option to determine whether the Responder has the necessary supportive Java class, prior to establishing the security channel. If the Responder currently does not hold the supportive Java class (i.e. the Responder is incompatible to execute the Java code from the Initiator), the Responder can make a request to the Initiator for the supportive Java class using the [EEr] option. Note that this is a simplified example, which involves one support class in Java; in reality, active code execution may require much more supportive information.

The use of the [EEi] and [EEr] options in SPAN improves robustness because - at the time of writing - in current active networking systems, there is no provisioning to accommodate incompatibility between active nodes. Thus, existing active networking systems must assume compatibility between *all* active nodes. By compatibility, we mean that the management application is assumed capable of creating static code that is guaranteed to be executable on all remote nodes. This implies that such systems would easily fail to operate if deployed in practice because of the high degree of heterogeneity in a large network such as the Internet. However, in SPAN, an Initiator can now make *authenticated* and *integrity protected* queries on remote node/EE's compatibility, and receives *protected* replies, *prior to* the actual hop-by-hop SA establishment and active packet transmission. The Initiator either can therefore ignore an incompatible node, or provides the necessary adjustments to ensure static code compatibility on remote Responder (such as supplying the Responder with a missing module that is required to execute the packet on the Responder). Because the query process is conducted at the initial stage of the SA establishment process, incompatibility between nodes would have been

identified (and resolved, if possible) prior to any other further processing (such as SA establishment). An incompatible Responder will not respond to the Initiator, but will forward the SPAN_INIT message to its neighbours (neighbours other than the Initiator), where the protocol may potentially continue. Robustness of the underlying active networking systems is therefore enhanced, because provisioning is made in the protocol to accommodate incompatibilities between nodes. The overall efficiency of active networking systems is also enhanced, in the sense that communicating nodes must ensure compatibility *prior to* any other further communications or processing (i.e. not wasting resources on establishing SAs with incompatible nodes).

4.5.2 Enhancing Flexibility in SPAN

Besides provisioning for static code hop-by-hop negotiation between intermediate nodes (i.e. through [EEi] and [EEr]), SPAN enhances flexibility by enabling SA negotiations between nodes. By using the SAi and SAr fields, nodes can negotiate security parameters (e.g. supported/preferred encryption algorithms and key size) in a hop-by-hop manner. It should be noted SANTS does not address hop-by-hop key establishment; whereas pre-distributed shared key, SKT, and SANE do not support SA negotiation (and EE query). As such, these (related) approaches have limited flexibility when deployed in a heterogeneous environment because individual security needs may not be satisfied. On the other hand, SPAN, IKE+IPSec, and KSV support SA negotiation; and an asymmetric approach (i.e. digitally signing and encrypting packets) may be deployed in heterogeneous networks (as long as PKI is supported). Thus, in the evaluation section, the efficiency and scalability of SPAN will be compared to IKEv2+IPSec, IKEv1 in aggressive mode+IPSec,

KSV, and asymmetric approach only.

4.5.3 Enhancing Scalability in SPAN

SPAN enables dynamic hop-by-hop SA establishment to be carried out by each individual node, thus SPAN scales better than centralised approaches such as KSV or SANE. SPAN is more scalable in the sense that each participating node needs to maintain the state of its *immediate* neighbours only (note that these neighbours do not necessary have to be a *physical* neighbouring node, but could be a neighbouring node on an *overlay* network). There is no need to maintain the state of the entire network (in KSV, a centralised keying server would have to maintain SAs for all registered nodes; whereas in SANE, some workarounds require the source node to establish trust with each of the nodes through which an (active) packet has traversed). Thus, in SPAN, the number of states maintained on one node is not dependent on the size of the network, but rather on the number of neighbouring nodes only.

4.5.4 Summary

- SPAN enhances robustness in active networking systems by enabling peers to use secured messages to ensure compatibility, prior to actually establishing secured tunnels and sending across active packets.
- Flexibility in SPAN is achieved by enabling peers to negotiate cryptographic algorithms prior to establishing secured tunnels.
- Scalability in SPAN is achieved through dynamic compatibility and security negotiations; also, SPAN is decentralised: each node may establish tunnels with each other.

4.6 Efficient Detection of DoS Attacks

Key exchange protocols are subjected to DoS attacks. There have been several reports [70][71] on DoS attacks on IKEv1 (which is a *standardised* key exchange protocol). For example, a DoS attacker can either create (large number of) legitimate key establishment instantiation requests, in an attempt to overload a Responder [72]; or it can flood a Responder with initialisation requests with forged IP addresses; or it can randomly modify the payload of a legitimate request message, causing a cache miss at a Responder[10]. In this section, we will discuss how SPAN addresses DoS attacks.

4.6.1 DoS Attacks in IKEv2

DoS attacks are possible in IKEv2. For example, an IKEv2 Responder - upon receiving an initialisation message (i.e. IKEv2 message 1) from an attacker - would be wasting computational resources. It will need to create (new) D-H values (for IKEv2 message 2), compute shared key sets (upon receiving valid/invalid IKEv2 message 3 from the Initiator), and will try to decrypt the encrypted IKEv2 message 3 from the Initiator, prior to verifying the authenticity of the Initiator[11]. The same problem is experienced in JFK. A JFK Responder - upon receiving the first message from the Initiator (in this case an attacker) - would be wasting resources on computing (new) D-H exponentials, creating digital signature over the D-H exponentials, creating a keyed hash over keying materials, computing the shared key set (upon receiving JFKi/r message 3),

[10] One form of DoS attacks is that an attacker randomly changes the contents of a message. When the Responder receives the modified message, the integrity check on the message will fail. More specifically, the hash value (of the modified message) computed by the Responder (which is temporally stored in the Responder's cache memory) does not match with the hash value that was included in the message. An attacker can flood the Responder with malformed messages, forcing the Responder to waste resources.

[11] In IKEv2, the Initiator's only signature i.e. AUTH on keying materials is kept in an *encrypted* payload in message 3. Thus, the Responder must compute the shared key set, prior to being able to verify the Initiator's signature.

and trying to decrypt the encrypted contents in JFKi/r message 3, prior to verifying the Initiator's digital signature[12].

4.6.2 IKEv2 Defence Mechanisms for DoS Attacks

A variant of IKEv2 includes the use of COOKIEs to implement limited DoS protection on participating nodes (section 2.10.4 on p.64). This variant of IKEv2 involves an exchange of six messages (rather than the standard four), and the IKEv2 Responder verifies the identity of the Initiator in the *fifth* message in its exchange. The term "COOKIE" originates in Photuris [73], which was an early proposal for key management with IPSec (which is now replaced by IKEv2). In a variant of IKEv2, a COOKIE is a randomly generated piece of data that is used for addressing DoS attacks (section 8.3 on p.164).

In brief, a Responder is configured to reject initialisation requests, and responds to the Initiator with an unprotected message that contains a COOKIE. The IKEv2 Initiator must then *resend* the initialisation message with the valid COOKIE to prove that it is using the same IP address as the one used in the (rejected) first initialisation message.

4.6.3 DoS Attacks on IKEv2 with COOKIEs

The IKEv2 RFC claims this arrangement can be used to implement limited protection against DoS attacks. In the IKEv2 RFC, it was discussed that this arrangement would enable the protocol to start with a weaker form of authentication (of IP addresses), and possibly later performing stronger

[12] JFK suggests a mechanism to address DoS attacks by requiring the Responder to periodically generate D-H exponential tuples (every 30s), and use a First-In-First-Out (FIFO) approach for assigning D-H exponentials to Initiator's requests; but this arrangement would add overhead (i.e. state maintenance) to the Responder even when there is no attempt from other nodes to establish hop-by-hop SA; more importantly, a JFK Responder would still be wasting resources for all other computational expensive processes e.g. computing signature and computing shared key set, prior to detecting DoS attacks (as explained in the main text).

authentication. However, the author of this thesis argues that the use of COOKIEs in IKEv2 does not provide a complete solution to the problem [72]: this is because attackers can intercept and modify all messages from the Responder (assumed). Thus, an attacker can generate the first initialisation message using a valid IP address of another node (i.e. a victim node). The Responder responds by sending the reply (i.e. a COOKIE) to the victim node, using the victim node's legitimate IP address as the reply's destination address. The attacker then intercepts the COOKIE from the reply sent by the Responder to the victim node, and (re)sends the same initialisation message with the valid COOKIE. The attacker puts the victim node's IP address as the source address of his/her message. The IKEv2 Responder is tricked to believe the IP address used by the Initiator does belong to the Initiator (because the attacker appears to be re-sending the valid COOKIE from the same IP address as the first initialisation message); and carries out all the computationally expensive processes (i.e. generating D-H values or computing share key set). More importantly, the Responder is unable to distinguish legitimate request messages from DoS attacks until a much later stage: because until receiving IKEv2 message 5 (which contains the Initiator's signature on keying materials), the IKEv2 Responder still believes the Initiator is a legitimate requester. The use of COOKIEs, however, would enable the Responder to identify that the attacker has access to physical link on the route from the Responder to the victim's IP address (i.e. the spoofed IP address being used by the attacker). As discussed in [72], in reality, the attacker is probably quite "close" to either the Responder or the victim node. Thus, the use of COOKIEs makes tracing easier.

4.6.4 SPAN Defence Mechanism for DoS Attacks

DoS attack is an important issue to be considered when designing practical key exchange protocols: as discussed in [72], detection of DoS attack is the *fundamental* countermeasure against DoS attacks; as such, countermeasures are introduced in SPAN against DoS attacks by limiting the resources required by a SPAN Responder to identify legitimate requests from DoS request messages [57][67][72]. The first countermeasure of SPAN against DoS attacks is that a SPAN Responder may carry out *essential* operations only until it verifies whether the request is a legitimate request or part of a DoS attack. As discussed in the assumptions, all authenticated request messages (i.e. with valid digital signatures) are assumed legitimate requests; whereas others are attack messages. Therefore, DoS attack requests can be distinguished from legitimate requests by requiring the Responder to verify the authenticity of the requests. The approach (of verifying the identity of a peer prior to any further communications) to counter DoS attacks was also proposed in [57]; but SPAN aims to achieve this goal much more rapidly.

Note that SPAN is capable of verifying the Initiator's authenticity when the Responder receives the *first* message in the protocol i.e. SPAN_INIT. This is because SPAN requires the Initiator to digitally sign SPAN_INIT using its valid PKI private key. If the signature on SPAN_INIT cannot be verified, the SPAN Responder will consider the message is an attack message, and the SPAN Responder will not proceed further. Thus, SPAN can more quickly identify legitimate requests from DoS attacks than existing approaches such as JFK.

To address another form of DoS attack that involves random modification of encrypted contents of duplicated SPAN_AP, the SPAN Responder caches the

corresponding SPI values of SPAN_AP. Thus, a duplicated (and/or malformed) SPAN_AP can be quickly detected and dropped by the Responder simply by matching SPI values in the message header and those values in its cache.

4.6.5 Discussion of SPAN's Anti-DoS Mechanisms

Although one may argue this would not completely eliminate DoS attacks, i.e. it still requires some resources to detect DoS attacks, but the author argues that:

1. This is the only arrangement that would enable the Responder to distinguish legitimate request messages from DoS attacks at the *very first stage* of the protocol exchange *prior to any other computationally expensive processes* such as D-H exponential computations and shared key set computations.
2. The rule-of-thumb in key exchange design is to *reduce* the impact of DoS attacks: in the evaluation section, it is proven that the SPAN approach enables much more *rapid detection of DoS attacks* than existing approaches i.e. less impact on the Responder.

Note that the cost of signature verification (at the Responder) can be reduced or even neglected [74] by using carefully selected parameters for asymmetric algorithms: for example use a relatively small public exponent e (but larger values for secret prime numbers p and q) [75] to achieve a quicker RSA signature verification[13]. A discussion of the performance of signature verification can be found in section 8.14.5 (p.188). It was discussed in [76][77] that with careful selection of parameters, the performance of RSA can be improved without lowering the level of security of the protocol[14].

---

[13] In brief, the RSA signature verification process is to raise the signature to the power of e mod n. A small value of e therefore reduces the computational cost for computing the exponential, hence achieving more efficient signature verification.

[14] Attacks on a message *encrypted* by using low-exponent RSA was identified [76], which enables the recovery of the plaintext. The corresponding defence mechanism (through padding

4.6.6 Summary

- DoS attack must be addressed in key exchange protocols.
- IKEv2 uses COOKIEs to address DoS attacks.
- A weakness of using COOKIEs to defend against DoS attacks in IKEv2 is discussed.
- SPAN addresses DoS attacks by requiring communicating peers to verify each other's authenticity, prior to carrying out any further computational process.
- Asymmetric techniques are used to achieve anti-DoS mechanisms in SPAN. Techniques for reducing overhead are addressed.

4.7 The Use of Asymmetric Cryptography in SPAN

Although asymmetric cryptography can be used to protect active packets, it is performance costly to digitally sign and encrypt *every* packet. Generally, asymmetric cryptographic techniques are used to sign symmetric keys, and the symmetric keys are used for packet protection. For example, SKT requires a node to digitally sign a symmetric key, and sends the signed key to other nodes in the network. Subsequent packets are protected by using the symmetric key. Asymmetric cryptography, however, is a widely acceptable technique to provide non-repudiation protection. An example is the PKI infrastructure. Given that active packets may carry executable *control* or *management* code that is to be executed on remote nodes, non-repudiation protection on packets' code is essential. Thus, there is a need to find a *balance* point between efficiency and strong security.

---

with nonce) has been identified [106]. Readers should note that this does *not* affect the use of low-exponent RSA in SPAN because we use RSA for *signatures*, rather than encryption.

In SPAN, the use of asymmetric cryptography is kept to a *minimum*. It is used for *essential* non-repudiation protection only, i.e. signing static control code of active packets, and for verifying *initial* exchanged keying materials only. It is not practical to create AUTH payloads by using shared secret as a pre-shared key (i.e. MAC) as suggested in the IKEv2 RFC. This is because this approach falls into a chicken-and-egg problem: to establish a shared key one must first have a shared key.

Note that the SPAN protocol requires the static code of an active packet in SPAN_AP to be signed *separately* from the AUTHi payload. This is because, the static code signature must be a separate signature (separated from other signatures created during the SPAN protocol exchange) so that the Responder – once becomes a new Initiator - can *append* this signature to the new active packet for source authenticity and integrity protection on the static code. The private key used to sign static code should belong to the principal (i.e. the actual creator of the code on the originating node); whereas AUTH should be signed by using the node's private key. This arrangement of using two private keys is to provide *strong* non-repudiation protection. The static code on an active packet was created by the principal e.g. an administrator/management application, so the principal's private key should be used to sign static code; whereas the keying materials were created by the node, so the node's private key should be used to sign them.

However, if a principal does not have a public key pair to sign static code, the node's (that the principal is currently residing) private key could be used instead. These arrangements enhance scalability (less PKI key pair required in the network) at the expense of less than ideal non-repudiation protection (i.e. the

code is now signed by the node *on behalf of* the actual creator). SPAN neither restricts static code to be signed by the principal, nor requires each node and each administrator/management application to be equipped with its public key pair.

There are some scalability concerns on maintaining potentially large Certificate Revocation Lists (CRLs) (to support signature verification). Digitally signed static code using a PKI private key can be easily verified on a node if the node has been equipped with the same PKI key pair (because the node would already have the corresponding certificate of the source node's/principal's PKI key pair). Otherwise, the CRL problem exists. One solution would be to configure nodes to download CRLs when network use is low; but this requires additional monitoring tool in place to determine real-time network traffic. A more practical solution would be to configure nodes to download CRLs at random times to avoid bursts of traffic [78]. Readers should note that it is *essential* to digitally sign static code using a standardised, common technique such as PKI. This is because some level of non-repudiation protection must be enforced on static code. Note that static code are executed for *management* or *control* purposes, so sophisticated protections must be in place so that the principal (i.e. the Initiator) cannot deny of deploying the (potentially damaging) code. Currently, asymmetric cryptography (such as PKI) is the only candidate technique that has been widely used (e.g. embedded in all web browsers) to support non-repudiation protection. Our design neither restricts static code to be signed by the principal, nor requires each node and each node/administrator/management application to be equipped with its public key pair. The choice of whether relying on the node to sign static code, or to equip

each node/administrator/management application with their own public key pairs, should be made by the administrators or the SPs that operate the active networking systems i.e. depending on the level of non-repudiation protection that is desirable to the administrators/SPs.

4.7.1 Summary

- The use of asymmetric cryptography in key exchange protocol is unavoidable due to the need for non-repudiation protection.
- However, asymmetric cryptography creates more overhead than symmetric cryptography.
- SPAN has minimised the use of overhead by using asymmetric cryptography only when necessary (i.e. for creating signatures on keying elements and static code).

4.8 Applicability of SPAN

SPAN is applicable whenever a new hop-by-hop SA is required i.e. between participating nodes along a *new* execution path. One may argue that SPAN would have limited applicability when pre-established SAs exist. Note that this is not a fair argument because this argument is based on the assumption that hop-by-hop SAs *pre-exist*, whereas SPAN is to *establish* hop-by-hop SAs. Also, as discussed in the assumption section (section 3.1 on p.74), currently there is no requirement for all nodes in the entire Internet to have *pre-established* trust with each other.

One concern would be that there is a chance that an active packet is sent along an execution path along which no SA has been established or an established SA has expired. In a small-scale network such as a LAN, it is possible that shared keys have already been pre-established or constantly being renewed

between all nodes; but this statement cannot be applied to a large-scale network, especially when active packets support dynamic routing. This is because, as explained in an earlier section, the next hop of execution does not necessary have to be to a physical neighbouring node within the same administrative domain, but could be to *any* active node on the *heterogeneous* large Internet. Thus, an efficient hop-by-hop SA establishment and a remote EE information query protocol such as SPAN is needed.

One may further argue that SPAN would have limited applicability because pre-established trust could be assumed *within an administrative domain.* In this case, SPAN is deployed between gateways of heterogeneous domains only. This statement could be valid if the administrative domain is *small* or *homogeneous*: that the administrator/management application of the domain has sufficient knowledge of his domain, and is able to equip each pair of hops with *different* keys. Different keys are needed because if more than one pair of hops share the same key, then per-hop authentication would fail (this is essentially the same problem in multicast IPSec). However, if the administrative domain were large or heterogeneous, an automated tool such as SPAN that generates much less performance overhead than traditional techniques (see later on evaluation) would be better suited for creating hop-by-hop SA within the domain. In *both case, the opportunity of an active packet being sent across heterogeneous administrative domains, and* subsequently the applicability of SPAN, would depend on the scale of deployment of the underlying active networking systems. The wider the scale of deployment of active networking systems, the more heterogeneous administrative domains/nodes would be involved, and as such the higher the opportunity an active packet will traverse

heterogeneous domains/nodes. Thus, an efficient, flexible, and scalable hop-by-hop security approach such as SPAN is needed for large-scale deployment of active networking systems.

4.8.1 Summary

- SPAN is designed for establishing new SAs between nodes; it may also be used to re-establish expired SAs between nodes.
- The use of SPAN in an environment in which pre-established trust exists is addressed.
- SPAN may be used in homogeneous or heterogeneous environment (or wherever an automated, scalable key exchange tool is needed).

5 Evaluation

In this chapter, SPAN will be evaluated against some of the existing protocols by its efficiency, scalability, behaviour under DoS, and flexibility. The choice of existing solutions used in this evaluation (for comparison against SPAN) will be explained.

For efficiency evaluation, the total time required by SPAN and different variants of IKEv2+IPSec to complete one protocol handshake and to transmit an active packet between a pair of nodes is determined. This figure will enable the readers to notice the efficiency difference between SPAN and different variants of IKEv2. For scalability evaluation, the effects of deploying SPAN and other existing protocols in a simulated large-scale network will be determined. Different implementations of DoS attacks are launched against the protocol, to determine the differences between the behaviour of each protocol under DoS attacks. For robustness and flexibility, SPAN is evaluated against existing solutions in terms of how it accommodates incompatibility and heterogeneities in networks.

5.1 Packet Format Designs

The SPAN protocol consists of three messages: SPAN_INIT, SPAN_AUTH, and SPAN_AP. The packet format designs for these messages are presented in this section.

5.1.1 An Overview on Packet Format Design

SPAN packets may be encapsulated into the payload of a TCP or UDP packet. Figure 34 shows a generic representation of a SPAN packet when encapsulated into the payload of an UDP packet.

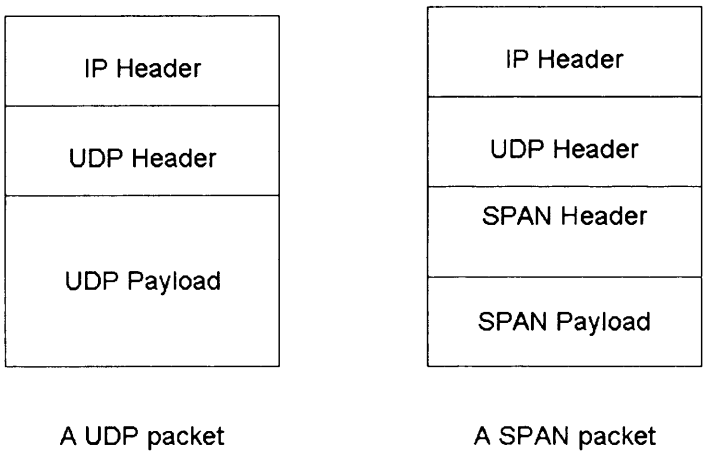

Figure 34 – A generic representation of a SPAN packet

Each SPAN packet contains a SPAN message. To identify a SPAN message, each SPAN message must be defined with an appropriate Exchange Type. An Exchange Type is a value placed in the SPAN header, which allows the receiver to identify the type of the message (i.e. whether the received message is a SPAN_INIT, SPAN_AUTH, or SPAN_AP). To avoid potential complications with standard IKEv2 messages, a set of private values should be used as the Exchange Type of our SPAN messages. SPAN_INIT uses 240, SPAN_AUTH uses 241, and SPAN_AP uses 242. Similarly, SPAN-specific payloads are identified using specific values for the Payload Types. SPAN-specific Payload Types (e.g. [EEi] and [EEr].) are specified by using private values in-between 128-255.

5.1.2 A Generic Packet Format Design for SPAN_INIT

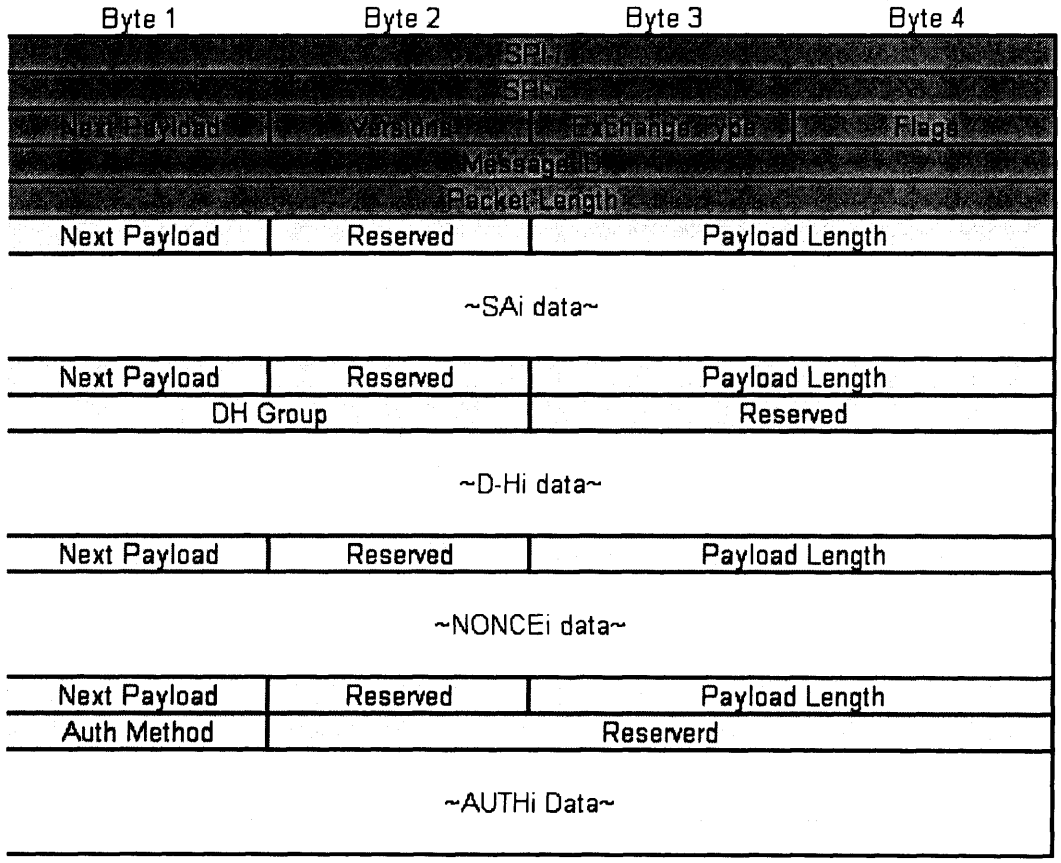

Figure 35 – Packet format for SPAN_INIT

Figure 35 shows a generic packet format for a SPAN_INIT message. This packet format contains all the fields needed to carry the compulsory elements in the SPAN INIT message defined in section 3.4 (p.78). The Exchange Type field tells the receiver this message is a SPAN_INIT message. Each field holds one element in the SPAN INIT message, that is, the SA offered by the Initiator to the Responder (i.e. SAi), the Diffie-Hellman public values offered by the Initiator to the Responder (i.e. D-Hi), the Initiator's nonce (i.e. NONCEi), and the Initiator's digital signature of the message (i.e. AUTHi). The SPAN header contains references to the SPIs that will be used by the Initiator (i.e. SPIi) and the

Responder (i.e. SPIr, but zero for SPIr because this field must be filled in by the Responder in the SPAN_AUTH message only). The Next Payload field enables the packet receiver to determine what to be expected immediately following the SPAN header (in this case it is the SA from the Initiator). The Message ID field keeps a dummy message ID for this message. The protocol exchange starts with message ID 1. The ID is incremented each time a message is successfully received. For example, in the first exchange, a message with ID 1 is sent from the Initiator to the Responder. The Responder sends a message with ID 2 back to the Initiator, and so on. The message ID is set to zero when the SA expires.

5.1.3 Generic Packet Format Designs for SPAN_AUTH and SPAN_AP

These messages have a similar format to SPAN_INIT; thus, only the differences will be discussed in detail in this section.

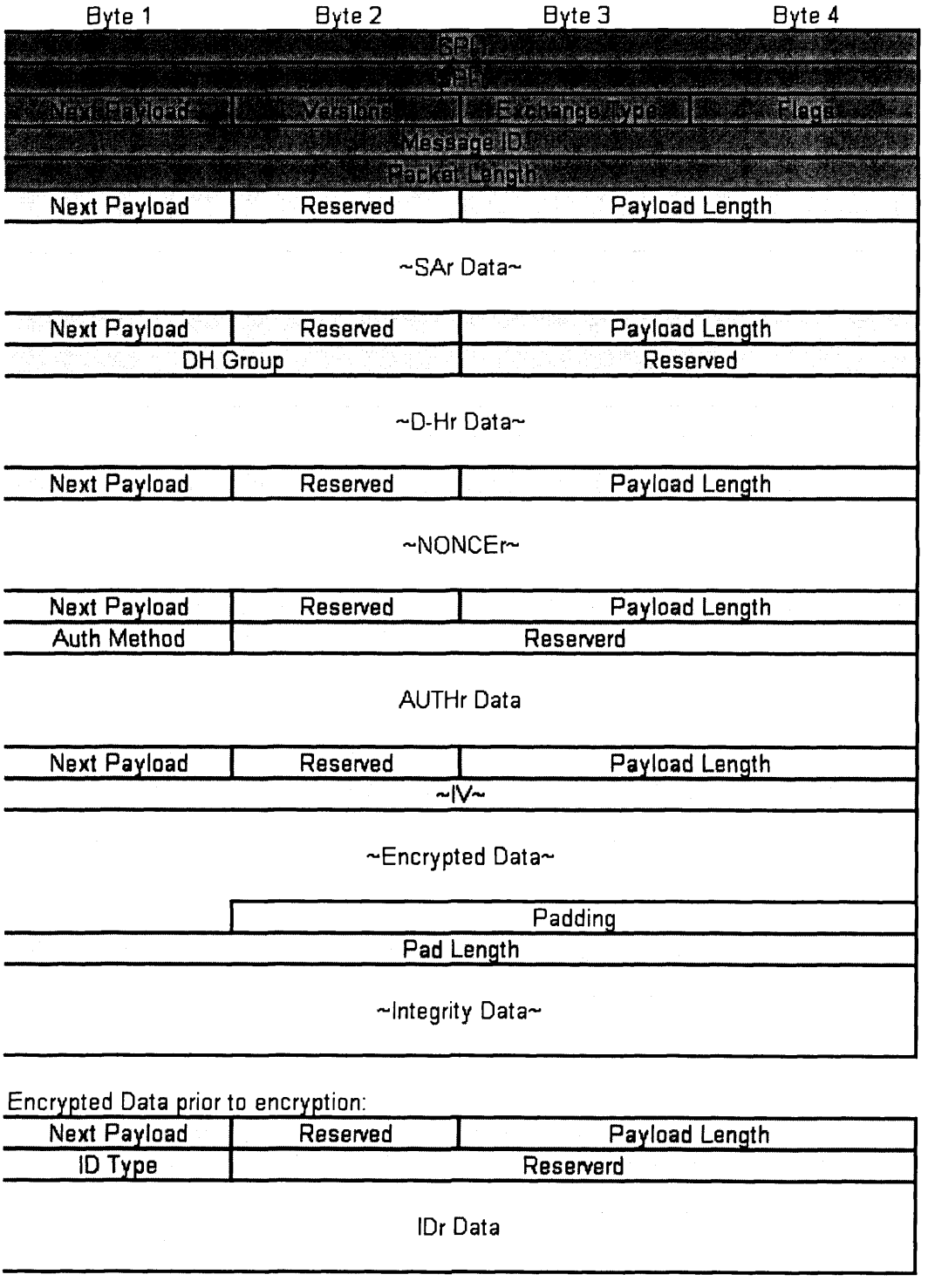

Figure 36 – Packet format for SPAN_AUTH

Figure 36 shows the packet format for SPAN_AUTH. It shows all the fields needed to contain the elements in SPAN_AUTH, which are defined in section

3.5 (p.82). Note that the Responder will fill in the SPIr field, using a value that it refers to its (to-be-established) SA. The message ID is incremented. The AUTHr field keeps a digital signature, which is created by using the Responder's private key. The signature covers the header, SAr, D-Hr, NONCEi and NONCEr. Note further that the Encrypted Data field contains the encrypted Responder's ID (i.e. IDr). The Initialisation Vector (IV) field keeps an IV for the encrypted payload. This value would be needed in order to address a security weakness in some modern encryption ciphers. This weakness is addressed by adding a level of randomness (i.e. by using the IV) to the encryption key each time the key is used for encryption. Detailed discussion of the use of IV in encryption cipher is an issue of cryptographic algorithm, which is out-of-scope of this thesis (see section 8.8 on p.177 for more detail). The encrypted data is appended by a checksum. The checksum is created using the freshly created shared key set, and it covers the encrypted payload for authenticity and integrity protection.

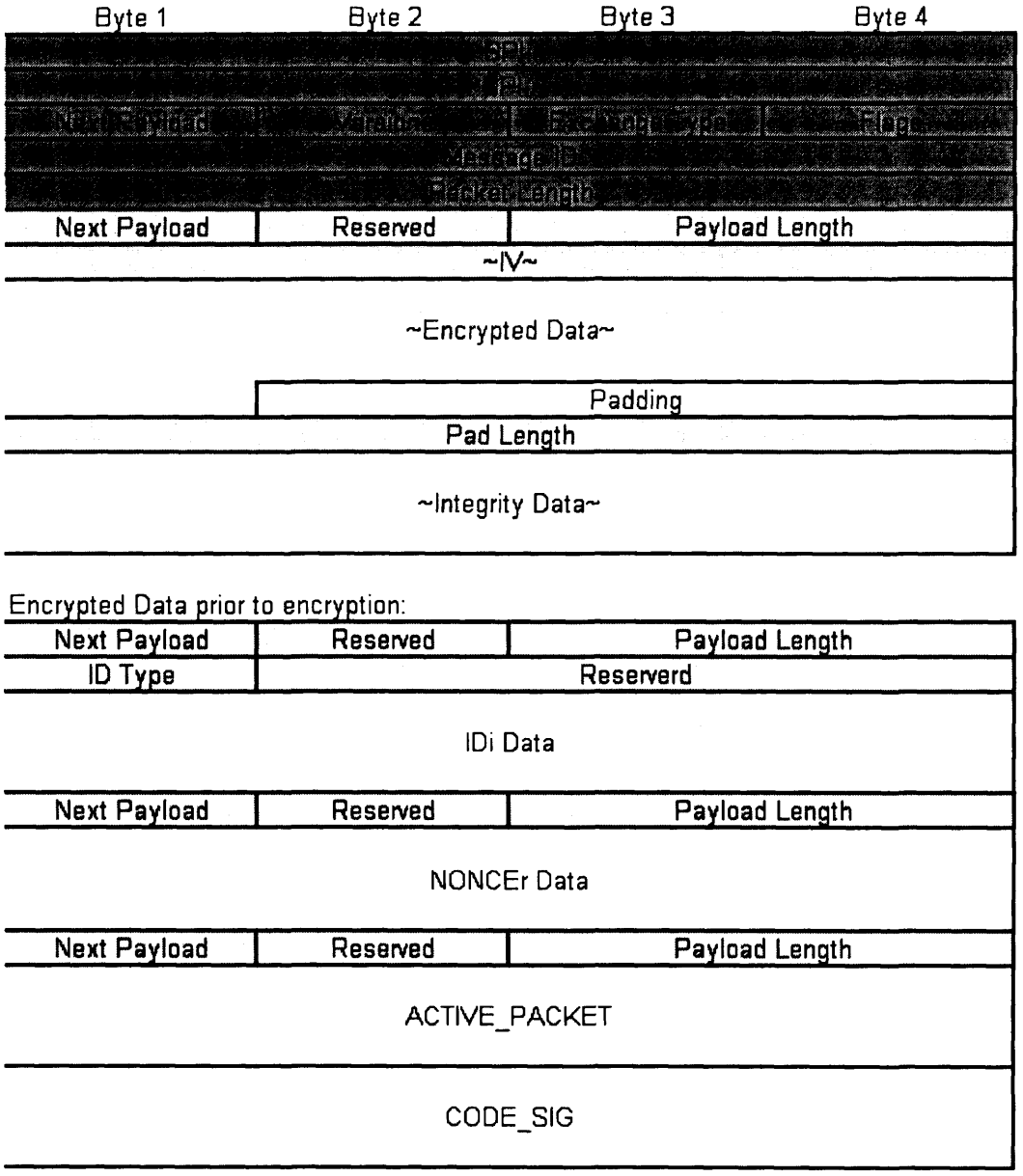

Figure 37 – Packet format for SPAN_AP

Figure 37 shows the packet format for SPAN_AP. It is similar to SPAN_INIT and SPAN_AP, except that it contains an encrypted payload only. The encrypted payload contains encrypted IDi, NONCEr, ACTIVE_PACKET, and CODE_SIG. CODE_SIG is the digital signature on the static code, it is created by the principal (i.e. which is also the Initiator in this case). The encrypted payload is appended by a keyed checksum (which is created using the freshly

created shared key set) for authenticity and integrity protection of the entire message.

5.2 Experiment Setup

For the simplest hop-by-hop evaluation, two interconnected nodes are needed. One node should be used as the Initiator and the principal (i.e. that an active packet is about to be injected), the other node should be used as the Responder (i.e. that the active packet will be received). This arrangement simulates a hop-by-hop environment. In practise, however, a hop-by-hop transmission can involve several nodes, with intermediate nodes being passive routers that simply intercept-and-forward IP packet.

The protocols under evaluation were run between two laptops (each with an Intel Pentium M processor 1.70GHz, 796MHz cache, with 1GB RAM). The two laptops were connected through an Ethernet cable. The machines have a shared directory for keeping common data. Linux (with JDK1.5) was installed on these machines. Figure 38 shows the experiment setup for evaluating the SPAN protocol.

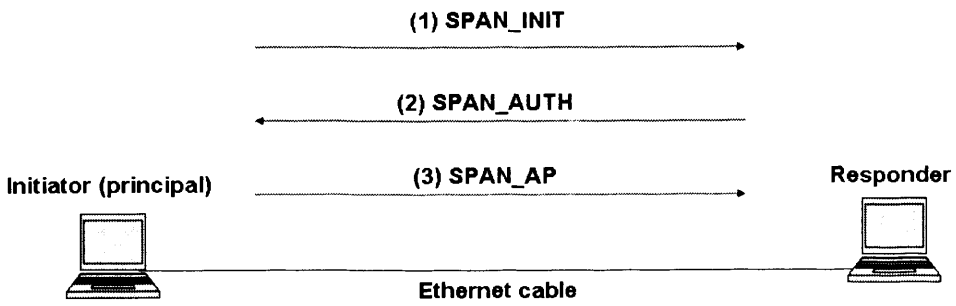

Figure 38 – Experiment arrangement

5.3 Prototype Design and Implementation

In this section, the prototypes are described. Their design choices are explained. Note that the prototypes are implemented for protocol evaluation only. The protocols are not limited to the techniques that were chosen to implement the protocol prototypes.

5.3.1 Choosing Programming Language

For a fair evaluation, the prototypes of the protocols under evaluation must be developed using the same programming language. This is because different programming languages have different performance (e.g. C is faster than Java). A prototype of SPAN and the relevant components of IKEv2[15] were developed

[15] The reason for developing (relevant parts of) IKEv2 is that at the time when the implementation starts (early 2005), no open source of IKEv2 was available.

in Java for evaluation. Java was chosen because of its compatibility and its usability.

5.3.2 Choosing Cryptographic Algorithms

Again, for a fair evaluation, the prototypes must use the same set of cryptographic algorithm. This is because different cryptographic algorithms have different performance e.g. Triple Data Encryption Standard (TDES) would be less efficient than Data Encryption Standard (DES). Generally, the more sophisticated the cryptographic algorithm, the more resources it would need; but relatively, they are more secured. However, it should be noted that this evaluation is not to evaluate levels of security of a chosen cryptographic algorithm; but to evaluate the efficiency between the chosen key exchange protocols. Thus, the chosen cryptographic algorithm(s) should be easy to implement. DES and DSA are chosen as the encryption algorithm and digital signature algorithm. The choices will be explained shortly later.

5.3.3 The SPAN Package

The SPAN package is designed for evaluation purpose; as such, simplicity is the main design criteria. As indicated, it is implemented in Java. An overview of the package's class files is provided below. The code that implements the classes will be discussed shortly afterwards.

- The SPAN_R class

This is the main class that implements the SPAN Responder. It has 363 lines of code. This class is started first and listens on a port for incoming initialisation request (i.e. SPAN_INIT) from remote SPAN Initiators. It verifies the signature of the initialisation request, computes its own D-H values, and computes the shared key set. It then prepares the SPAN_AUTH message, and sends it back

to the Initiator through a socket. Lastly, it waits for the Initiator's last message (i.e. SPAN_AP).

- The SPAN_I class

This is the main class that implements the SPAN Initiator. It has 381 lines of code. This class starts the SPAN protocol by generating a D-H value, creates a signature of the message (i.e. AUTHi), and sends the SPAN_INIT message to the Responder. It then intercepts the SPAN_AUTH message from the Responder, verifies the message's signature, and computes the shared key set. It sends the last message in the protocol, i.e. SPAN_AP, to the Responder. The message includes a dummy active packet, i.e. ACTIVE_CODE, which subsequently includes the active packet's static code (i.e. STATIC_CODE). The size of the active packet and its static code is variable.

- The KeyDisplayer class, the KeyReader class, and the KeyWriter class

These classes are used to display the DES key (i.e. the shared key), and to read and write the DES key to local storage for future reference. They have 141 lines of code.

- The Signer class

This class is responsible for creating and verifying digital signatures in the SPAN protocol. It has 122 lines of code.

- .keystore

This file keeps the asymmetric key pair, which is needed for creating and verifying digital signatures in the SPAN protocol.

5.3.4 Creating and Verifying Digital Signatures in SPAN

In SPAN, digital signatures are created using Digital Signature Algorithm (DSA). DSA is used because it is a standardised technology for creating digital

signatures; also, it is supported in Java. DSA requires users to have pre-installed asymmetric keys (i.e. a keystore) for signing and verifying signatures. These keys are simulations of the public key certificates in the SPAN protocol. This requirement is acceptable for this evaluation because the SPAN protocol requires participating nodes to have public key certificates. This file is stored as a hidden file under the user's account. The keytool command in Linux is used to generate this key.

```
To generate a keystore file:
$ keytool -genkey -alias [alias name] -keyalg [key algorithm]
-keystore [keystore location]/.keystore
Example:
$ keytool -genkey -alias lcheng -keyalg DSA -keystore
/home/lcheng/.keystore
```

Figure 39 – Asymmetric key generation using keytool

To create and verify a digital signature, the Signer class is used. The Signer class contains methods for creating and verifying digital signatures, using the key pair information specified in the .keystore file.

```
// Assuming there is some data (i.e. raw_data) to be signed, and
// the signature will be stored (i.e. sig_file) for verification:
char[] storePassword = "12345678".toCharArray();
File sig_file = new File("data.sig");
String algorithm =
System.getProperty("signature.algorithm","DSA");
Signer mySigner = new Signer(algorithm, storePassword);
mySigner.createSignature(raw_data, sig_file, storePassword,
"lcheng");
...
// Assuming some signed data (i.e. raw_data) is to be verified:
ByteArrayInputStream in = new ByteArrayInputStream(raw_data);
File sig_file = new File("data.sig");
char[] storePassword = "12345678".toCharArray();
```

```
String algorithm = System.getProperty("signature.algorithm",
"DSA");
Signer mySigner = new Signer(algorithm, storePassword);
boolean valid = mySigner.verifySignature(in, sig_file, "lcheng");
```

Figure 40 – Creating and verifying a digital signature using the Signer class

Essentially, the key pair information from the .keystore file is read into memory. A Signer object is created, that is initialised with the chosen cryptographic algorithm (i.e. DSA), and the corresponding .keystore file password (so that the object has access right to the key pair information). The Signer object takes the data to be signed, extracts the private key from the .keystore file, and use the update() and sign() methods of the Signature class to create a digital signature of the data. The created signature is stored for verification at a later stage. Figure 41 shows the code of the createSignature() method of the Signer class.

```
public byte[] createSignature(byte[] inputTextBytes, char
keyPass[], String myAlias) throws GeneralSecurityException,
IOException
{
   PrivateKey pk = (PrivateKey)ks.getKey(myAlias, keyPass);
   sig.initSign(pk);
   byte[] buffer = inputTextBytes;
   int count = 0;
   for(int i = 0; i < buffer.length; ++i)
   {
      sig.update(buffer, 0, count);
   }
   byte[] signatureBlock = sig.sign();
   return signatureBlock;
}
```

Figure 41 – The createSignature() method of the Signer class

The signature verification process is similar. The Signer object takes the data

that was signed, the digital signature of the signed data, and extracts the public key from the keystore file, in order to verify the digital signature of the data. The method returns false if the verification fails. The code for the verifySignature is shown in Figure 42.

```
public boolean verifySignature(InputStream in, File sigfile,
String myAlias) throws GeneralSecurityException, IOException
{
   PublicKey pk = ks.getCertificate(myAlias).getPublicKey();
   sig.initVerify(pk);
   byte[] buffer = new byte[in.available() + 100];
   int count = 0;
   while((count = in.read(buffer)) > 0)
   {
      sig.update(buffer, 0, count);
   }
   in.close();
   FileInputStream signedIn = new FileInputStream(sigfile);
   byte[] signatureBlock = new byte[signedIn.available()];
   signedIn.read(signatureBlock);
   signedIn.close();
   return sig.verify(signatureBlock);
}
```

Figure 42 – The verifySignature() method of the Signer class

5.3.5 D-H Public Value Generation

```
DHParameterSpec myDHSpec;
AlgorithmParameterGenerator paramGen =
AlgorithmParameterGenerator.getInstance("DH");
paramGen.init(512);
AlgorithmParameters params = paramGen.generateParameters();
myDHSpec =
(DHParameterSpec)params.getParameterSpec(DHParameterSpec.class);
KeyPairGenerator myKeyPairGenerator =
KeyPairGenerator.getInstance("DH");
```

```
myKeyPairGenerator.initialize(myDHSpec);
KeyPair myKeyPair = myKeyPairGenerator.generateKeyPair();
KeyAgreement myKeyAgreement = KeyAgreement.getInstance("DH");
myKeyAgreement.init(myKeyPair.getPrivate());
byte[] iPublicDHValue = myKeyPair.getPublic().getEncoded();
```

Figure 43 – The code for creating the Initiator's D-H public value

The Exchange Type field tells the receiver this message is a SPAN_INIT message. Figure 43 shows the code for creating the Initiator's D-H public value, and encoding it, prior to sending it over to the Responder. First, an algorithm parameter specification is created. This specification is specified to use D-H as the public value generation algorithm. This specification, in turn, allows a KeyPairGenerator to be initialised. The KeyPairGenerator is the key component, which generates the D-H public and private values. It does this by calling the generateKeyPair() method. Note that the private output value of the D-H algorithm is kept locally at the Initiator, and never transmitted to the Responder. On the other hand, the D-H public value (i.e. iPublicDHValue) is sent to the Responder, so that the Responder can compute the shared key set. The D-H values generation process at the Responder is almost identical to the one shown above, except that the Responder's key pair is initialised using the Initiator's D-H public value (i.e. iPublicDHValue). Figure 44 shows the code at the Responder.

```
KeyFactory rKeyFactory = KeyFactory.getInstance("DH");
X509EncodedKeySpec x509KeySpec = new
X509EncodedKeySpec(iPublicDHValue);
PublicKey iPublicKey = rKeyFactory.generatePublic(x509KeySpec);
DHParameterSpec rDHSpec = ((DHPublicKey)iPublicKey).getParams();
KeyPairGenerator rKeyPairGenerator =
KeyPairGenerator.getInstance("DH");
```

```
rKeyPairGenerator.initialize(rDHSpec);
KeyPair rKeyPair = rKeyPairGenerator.generateKeyPair();
KeyAgreement rKeyAgreement = KeyAgreement.getInstance("DH");
rKeyAgreement.init(rKeyPair.getPrivate());
```

Figure 44 – The code for creating the Responder's D-H value

5.3.6 Shared DES Key Computation

Once the Initiator and the Responder has exchanged the D-H public parameters, they can establish the shared key set. The shared key set is calculated at the Responder as soon as the Responder has received, and verified, SPAN_INIT. The code shown in Figure 45 shows how the symmetric (shared) DES key is computed at the Responder, after it has received and verified the SPAN_INIT message from the Initiator.

```
rKeyAgreement.doPhase(iPublicKey, true);
SecretKey rDESKey = rKeyAgreement.generateSecret("DES");
byte[] rDESKeyBytes = rDESKey.getEncoded();
KeyDisplayer myKeyDisplayer = new KeyDisplayer();
String rDESKeyString = myKeyDisplayer.toHexString(rDESKeyBytes);
KeyWriter myKeyWriter = new KeyWriter("Responder.key", rDESKey);
myKeyWriter.writeToFile();
```

Figure 45 – The code for computing the shared key set at the Responder

The code shown in Figure 45 shows how the Responder calculates its symmetric DES key. The computation process needs the Initiator's D-H public value (i.e. iPublicKey). The Responder's DES key (i.e. rDESKey) is initially created as a SecretKey object, which is later converted into a byte array (i.e. rDESKeyBytes) for easy manipulation. The key is then further converted into a String using the toHexString() method of the KeyDisplayer class for display purpose. The original SecretKey object (i.e. rDESKey) is written to local storage

(i.e. Responder.key), so that it can be used to protect subsequent messages exchanged between the Responder and the Initiator. The Initiator uses the same algorithm to create its DES key.

5.3.7 Encryption and Decryption Code Implementation

Once the shared DES key is ready at both the Initiator and the Responder respectively, the key can be used to encrypt and decrypt data. All encryption uses DES in Electronic Code Book (ECB) mode. DES in ECB mode is used because it is a standardised, easy to implement technology, and it is supported in Java since Java 1.2. DES is extensible to support Triple DES (TDES), which is considered as a more secure encryption algorithm. ECB enables each possible block of plaintext to have a defined corresponding ciphertext; thus, using ECB enables the developer to check whether the encryption over a particular plaintext was carried out correctly. ECB mode supports PKCS5Padding, which adds dummy padding to the original (unencrypted) payload prior to encryption. This is essential process in encryption. Figure 46 shows the code used in SPAN for encryption and decryption.

```
// Assuming a DES key (i.e. myDESKey) is available and some data
// (i.e. cleartext) to be encrypted:

Cipher enCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
enCipher.init(Cipher.ENCRYPT_MODE, myDESKey);
byte[] ciphertext = enCipher.doFinal(cleartext);
...
// For decryption, use the same symmetric DES key (i.e. myDESKey):
Cipher deCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
deCipher.init(Cipher.DECRYPT_MODE, myDESKey);
byte[] cleartext = deCipher.doFinal(ciphertext);
```

Figure 46 – Encryption and decryption code in SPAN

The code essentially creates a Cipher object (i.e. enCipher), that is instantiated with the chosen cryptographic algorithms (i.e. DES, ECB, and PKCS5Padding) and the user's DES key. The user's DES key is created dynamically during the SPAN protocol (see section 5.3.6 on p.136). Then, the created Cipher object encrypts the data (i.e. cleartext) by calling the doFinal() method. The resultant encrypted data is in a byte array (i.e. ciphertext). The decryption process is asymmetric to the encryption process: a decryption Cipher object is created and instantiated with chosen cryptographic algorithms and corresponding DES key (i.e. deCipher). The decryption Cipher then decrypts the encrypted data (i.e. ciphertext) by using the same method (i.e. doFinal()), and returns the decrypted data in a byte array (i.e. cleartext).

5.3.8 Sending Packets on the Wire

It was discussed in section 3.10 (p.96) that packet lost in SPAN may be handled through traditional retransmission mechanism. For simplicity, SPAN messages are sent as UDP packets. Once a SPAN message is ready, it is sent to the other peer. The communication is via a DatagramSocket. Figure 47 shows the code used by the Initiator for sending the packet on the wire to the Responder, and listens for the Responder's incoming message.

```
// Send SPAN_INIT to Responder:

InetAddress rAddr = InetAddress.getByName(rAddrString);
sendSocket = new DatagramSocket(sendPort);
DatagramPacket packet = new DatagramPacket(SPAN_INIT_bytes,
SPAN_INIT_bytes.length, rAddr, rPort);
sendSocket.send(packet);
sendSocket.close();
...
// Now wait for Responder's message (i.e. SPAN_AUTH):
```

```
byte[] buffer = new byte[bufferSize];
packet = new DatagramPacket(buffer, buffer.length);
recvSocket = new DatagramSocket(recvPort);
recvSocket.receive(packet);
raw_data = packet.getData();
recvSocket.close();
```

Figure 47 – Packet transmission via DatagramSocket

Essentially, the Initiator opens a DatagramSocket (i.e. sendSocket), specifies the Responder's listening address and port (i.e. rAddr, rPort), and packages the SPAN_INIT message into a UDP packet (i.e. packet). It calls the send() method to send the packet to the Responder. Then, the Initiator opens another socket (i.e. recvSocket), and waits for the Responder's reply (i.e. the SPAN_AUTH message). When the reply arrives at the Initiator's listening socket, the data in the packet (i.e. packet) is extracted (i.e. raw_data) by the Initiator by calling the getData() method.

5.3.9 The IKEv2 Package

Although the design of the IKEv2+IPSec and SPAN protocols are different (IKEv2+IPSec uses six to eight messages to complete protocol exchange and active packet transmission, whereas SPAN uses only three messages), the two packages use the same classes and methods as described above. The reason for using the same classes and methods is because (as explained in section 5.3 on p.129), in order to establish a fair evaluation environment, the implementations of the protocols under evaluation must use the same technologies. For example, in IKEv2, the same createSignature() method presented in section 5.3.4 (p.131) is used to create digital signatures, and the same encryption method (section 5.3.7 on p.137) is used to encrypt data during

the key exchange protocol, and for encrypting active packets (i.e. a simulation of IPSec), and so on. The differences between the two protocols are not between the implementations, but are between the number of cryptographic processes (i.e. efficiency), and the number of messages (i.e. scalability), needed to complete the protocol. In the following sections, the efficiency and scalability differences between the two protocols will be discussed.

5.4 Efficiency and Scalability Evaluation

The performance of SPAN is compared with variants of IKEv2:

a) IKEv2+IPSec without PFS

b) IKEv2+IPSec with PFS support (new D-H values)

PFS is defined in [79][16]. PFS is optional [80] because it enables *strong* security [34], but incurs a high performance overhead because new D-H values are generated. Detailed discussion of PFS can be found in section 8.12 on p.180.

The IKEv2+IPSec approaches were chosen for efficiency assessment because:

1. The shared key *computation* process of SPAN is similar to the one used in IKEv2 (both uses D-H).
2. The IKEv2+IPSec approach is one of the approaches identified earlier in this thesis that provides a similar level of flexibility as SPAN, as they both support SA negotiation.
3. KSV essentially relies on IKEv1, and IKEv2 is meant to optimise IKEv1.
4. IKEv2 is a *standardised* protocol (published in the form of RFC) i.e. reasonable to be used as a benchmark for comparison.

The first measurement is efficiency, i.e. the time to complete one protocol

[16] PFS is defined as "*...an authenticated key exchange protocol provides PFS if disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs*" [79].

exchange (excluding packet execution time which is application-specific), is measured. In each trial, a dummy active packet (of 1024 bytes with a static code of 512 byte) is transmitted securely between the two peers during the SPAN protocol exchange. Note that it is the performance *differences* between different approaches that this thesis is measuring, not the actual performance results. This is because actual performance results are directly related to implementation and software design, which is a programming issue. Figure 48 shows the average results of 400 trials.

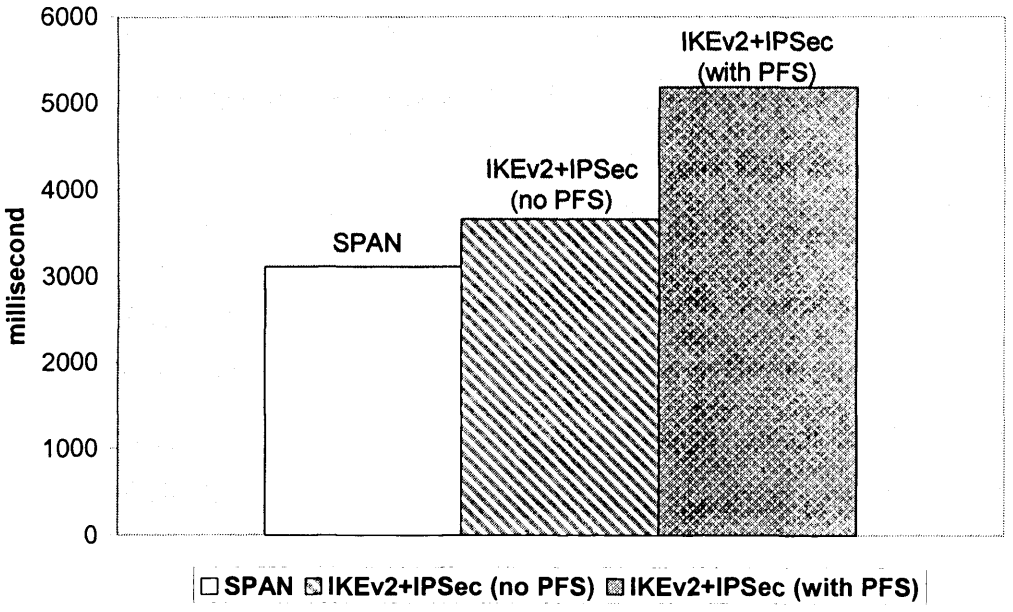

Figure 48 – Efficiency evaluation between SPAN and IKEv2 variants

The experiment results show that SPAN generates on average 15% to 40% less in performance overhead when compared to IKEv2+IPSec without/with PFS support. On average, SPAN needs 3103.76 milliseconds (±361ms), IKEv2+IPSec (no PFS) needs 3652.78 milliseconds (±303ms), and IKEv2+IPSec (with PFS) needs 5177.315 milliseconds (±366ms). The standard deviation shows the root mean square deviation of the values from their

arithmetic mean.

It is clear from the results that SPAN has a significant efficiency advantage over IKEv2+IPSec (with PFS) i.e. ~40% less in overhead; and a slightly less significant advantage over IKEv2+IPSec (without PFS) i.e. ~15%. Readers should note that:

- IKEv2+IPSec (without PFS) is more efficient than IKEv2+IPSec (with PFS) because IKEv2+IPSec (without PFS) achieves efficiency at the expense of less than ideal security. Although there has always been a challenge to determine a balance point between security and performance, in [80], it was stated that IKEv2+IPSec (with PFS) is ideal for situations where *strong* security is needed. Readers should note that active packets might carry *control* executable code, which could cause significant damage to systems if the code is compromised. Therefore, this thesis argues that strong security is desired in active network security solutions;
- It should be noted that one important factor is that SPAN is deployed in a *hop-by-hop* manner: this means the actual *magnitude* of the time delay reduction (i.e. actual time saved) *increases* as the scale of deployment of SPAN increases. This implies the efficiency advantage of SPAN - in terms of magnitude - becomes more significant as the scale of deployment of SPAN increases. Using the (average) results presented in section 5.4 (p.140), a 15% reduction in processing time of a process that originally takes 3600 milliseconds (i.e. IKEv2+IPSec without PFS) means the process would now take 3060 milliseconds only. One may argue that a 540 milliseconds efficiency advantage has very little impact. However, when deployed in large scale, say between 256 nodes (which is the typical

maximum TTL value in today's Internet), the magnitude of time reduction would be 255 x 540 = 137,700 milliseconds (2.295 mins). Similarly, when SPAN is deployed in a *large scale*, a 15% advantage becomes much more significant. More importantly, the significance of efficiency advantage of SPAN is directly related to the size of the deployment: *the larger the scale of deployment, the more significant the advantage*. This also implies that SPAN scales better than existing solutions (see later for more on scalability evaluation).

The results on efficiency are in-line with the arguments that were made in an earlier section. SPAN is more efficient because of the reduced exchange messages, i.e. reduced number of cryptographic operations on exchange messages (e.g. for message authenticity/integrity protection). Hence, SPAN has limited number of hashing, and requires only one D-H exponent generation (two are required in IKEv2+IPSec with PFS). One may argue IKEv1 in aggressive mode+IPSec may produce a similar performance result as SPAN due to its simplicity in key exchange handshake (section 2.12 on p.70): but as explained in an earlier section (section 4.4 on p.104), IKEv1 in aggressive mode+IPSec does not protect peers' identity (whereas identities are protected in SPAN); also, IKEv1 in aggressive mode requires more time to detect DoS attacks (section 5.5 on p.150). Furthermore, SPAN scales better than IKEv1 in aggressive mode+IPSec because SPAN enables secure active packet transmission *during* SA establishment; whereas IKEv1 in aggressive mode+IPSec can provide secure packet transmission only *after* IKE SA *and* IPSec SA establishment (section 5.4 on p.140).

One may argue that the use of asymmetric cryptography would be acceptable if

the number of active packets traversing a pair of nodes over a period (before the SA expires) is kept small. Thus, the *total time* to complete secure transmission(s) of different number of active packet(s) by using symmetric (i.e. SPAN) and asymmetric (i.e. DSA) approaches respectively over a period were measured. Note that each SPAN experiment included a dynamic establishment of a hop-by-hop SA; whereas no dynamic hop-by-hop SA was needed for deploying DSA because DSA uses asymmetric cryptography.

Figure 49 shows the results of SPAN vs. DSA, which shows the time needed for protecting different number of active packets across a pair of nodes (using SPAN and DSA respectively). For example, the total time needed to transmit three active packets across a pair of nodes using SPAN is 3221 milliseconds (±341ms), in contrast, DSA takes 2095 milliseconds (±78ms) only.

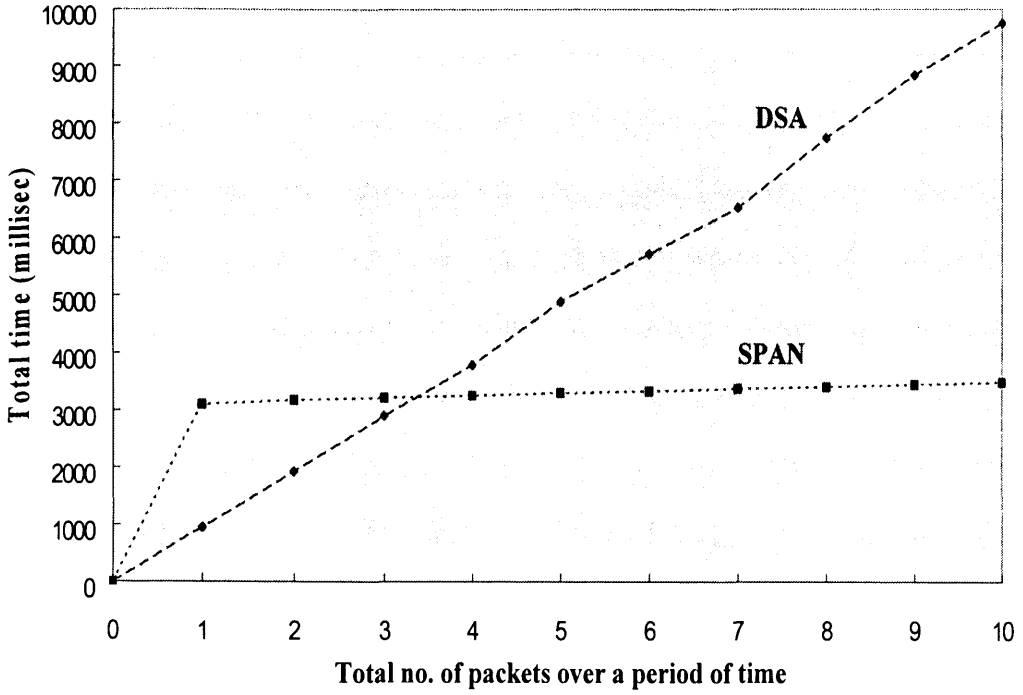

Figure 49 – Symmetric vs. asymmetric

The results (Figure 49) show that dynamic establishment of shared key set

does introduce more overhead than the asymmetric approach when the number of active packets being protected across a pair of nodes over a period of time is limited. However, the performance advantage of the asymmetric approach is overturned by SPAN when just four packets are to be secured over a hop over a period.

The results imply that symmetric approaches have scalability advantages over asymmetric approaches. This is because relatively much less time is needed to protect active packets by symmetric approaches when the number of active packets to be protected is large (i.e. large number of active packets to be protected means large-scale of deployment). The results shown in Figure 49 show that when protecting 10 active packets across a pair of nodes, SPAN takes 3491 milliseconds (±356ms), whereas DSA needs 9752 seconds (±125ms) i.e. SPAN has an efficiency advantage of ~64%.

One may argue that DSA still has the advantage when the number of packets to be protected is small. The reason why DSA has this advantage is that SPAN requires hop-by-hop SA establishment, which is a relatively time consuming process (the steep slope at the beginning of the curve of SPAN in Figure 49 shows the overhead for establishing a hop-by-hop SA). However, once the SA has been established, it can be re-used to protect subsequent active packets travelling along the same pair of hops. Thus, no SA has to be re-established (i.e. no time consuming process) until the SA expires.

Note that the standard lifetime of a SA ranges from 8 hours to 24 hours[17] [81]. The lifetime of a symmetric SA is much shorter than the lifetime of credential

[17] A SA must be replaced once it has expired. The expiry time is known as the lifetime of a SA. The author has carried out a search for the standard lifetime of a SA, but there is no single definition of a SA lifetime. The figures used in this thesis were identified in [81]: it was discussed that as a good practise, (symmetric cryptography based) VPN tunnels should be re-established between 8-24 hours.

references created by using asymmetric cryptography (e.g. Verisign are selling digital certificates that last between one to three years [82]). SAs are subjected to renewal because symmetric keys are less complex than asymmetric keys (Question 5 on p.146), and therefore must be replaced more frequently than asymmetric keys. This is because less complex keys are easier to be hacked. This implies that a hop-by-hop SA, once established, should remain for 8 to 24 hours. Thus, the author argues that, since active packets may be used for real-time control and management purposes, it is reasonable to assume that more than four active packets will traverse a pair of nodes over 8 to 24 hours. As such, the author argues that the SPAN approach (which uses a mixture of symmetric and asymmetric techniques) has an efficiency and scalability advantage when compared to approaches that use asymmetric techniques only.

Question 5: Explain the reasons why symmetric keys are less complex than asymmetric keys.

Symmetric keys are less complex because of the ways they are generated. According to the asymmetric key generation process described in section 8.14 (p.183) an asymmetric public key involves careful selection of large prime numbers. A resultant product of two large prime numbers is computationally impossible to factorise. Thus, asymmetric keys are difficult to hack. On the other hand, symmetric keys are generated randomly. For instance, by applying a secure hash function to a passphrase, and use the resultant hash value as the key [83]. Weaknesses in DES keys are discussed in [84].

Note that the results in Figure 49 also show that SPAN is more scalable than asymmetric cryptography, because the time delay for the asymmetric approach

increases much more significantly than SPAN as the number of packets protected increases (i.e. a steeper slope). This means the more the number of active packets to be protected, asymmetric approaches would take (proportionally) more time than symmetric approaches. This result is in-line with the discussion presented in Question 6 (p.166), that symmetric operations are much faster than asymmetric operations.

For further scalability evaluation, the following are compared:

(1) SPAN

(2) IKEv2+IPSec (with PFS)

(3) IKEv2+IPSec (without PFS)

(4) SANE

(5) IKEv1 in aggressive mode+IPSec

(6) IKEv2+COOKIE

(7) KSV

Scalability evaluation is conducted by determining the number of message exchanges required between peers in order to complete the protocols respectively along an execution path of 256 nodes. 256 nodes are chosen because that is the maximum TTL value i.e. to simulate large-scale deployment. TTL is the maximum number of hops that a packet is allowed to traverse. It is used as a technique to stop packets looping forever in the network.

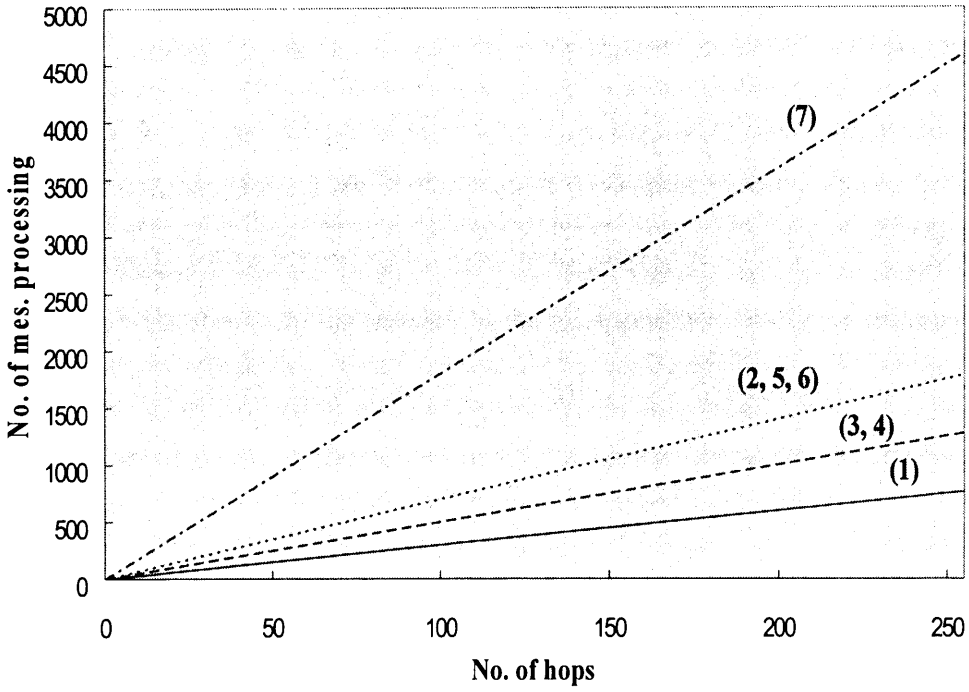

Figure 50 – No. of message exchanges along 255 hops

Figure 50 shows the number of message exchanges along a path of 255 hops under the different approaches. For example, SPAN requires three messages to be exchanged between a pair of nodes to complete one handshake; so for 255 hops, SPAN would need a total of 765 messages to be exchanged.

As shown in Figure 50, SPAN scales better than existing approaches. The larger the scale of deployment (i.e. more hops), the difference is more obvious. These promising results indicate that SPAN has an efficiency and scalability advantage over related work i.e. reduction in message processing and cryptographic operations and state maintenance.

Note that SPAN needs three messages to complete its protocol for a hop. One may argue whether there are ways to reduce further the number of message exchanges in SPAN. If dynamic key establishment is required, and if the rule of thumb "*...the communicating peers must be verified to each other (through*

verifying AUTHi and AUTHr) prior to active packet transmission" (section 4.2 on p.100) still applies, three is indeed the minimum number of message exchanges for per hop establishment. This is because one message must be reserved for sending across an active packet (note that active packet should only be sent across after the peers have *verified* each other's authenticity). One message must be sent from the Initiator to the Responder to verify the Initiator's authenticity, another message must then be sent from the Responder to the Initiator to verify the Responder's authenticity; note that these messages must be countersigned to combat impersonate attacks. Imagine a traveller purchasing a flight ticket from a travel agent. Prior to the actual transaction, the traveller must verify the authenticity of the travel agent e.g. by checking the agent's registered business licenses (i.e. the Initiator verifies the Responder). The travel agent must also verify the traveller's authenticity e.g. by checking the traveller's passport, credit cards... and more (i.e. the Responder verifies the Initiator), prior to completing the transaction e.g. handing over the flight ticket (i.e. the actual transmission of an active packet). Therefore, it is apparent that, three message exchanges per hop is indeed the least number.

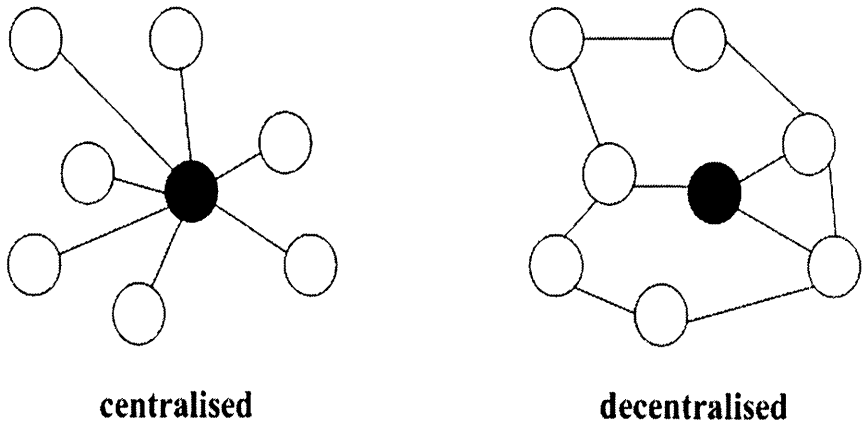

Figure 51 – Centralised approach vs. decentralised approach

Figure 51 explains the other important reason why SPAN is more scalable. This

is because SPAN (and others except KSV and some workarounds of SANE) is completely decentralised. More specifically, the number of state maintenance on a source node depends on the number of its immediate neighbours on a (physical or virtual) network. Under centralised approaches such as KSV and some workarounds in SANE, because a keying server/the source node is involved in hop-by-hop packet transmission/key establishment, the number of state maintenance of a source node depends on the scale of the entire (physical or virtual) network. Centralised approaches do not scale well in large-scale networks.

5.5 Evaluation on Detecting DoS Attacks

SPAN is compared against IKEv2 for evaluating the protocols under DoS attack resistance. IKEv2 is chosen because IKEv2 is a standardised protocol that has been used in many implementations. Furthermore, IKEv1 in aggressive mode+IPSec and JFK experience a similar DoS attack problem as IKEv2. This is because the first message from the (IKEv1 in aggressive mode or JFK) Initiator is not protected (i.e. authenticated) by any means; therefore, the (IKEv1 in aggressive mode or JFK) Responder carries out a series of cryptographic operations upon receiving un-authenticated or integrity checked initialisation messages. It will be discussed further shortly that this arrangement has a negative effect on DoS attack detection.

As discussed in an earlier section, IKEv2 uses a 6-message exchange with COOKIE as a countermeasure against DoS attacks (section 2.10.4 on p.64). Note that under this arrangement, an IKEv2 Responder does not verify the Initiator until the *fifth* message in its exchange. The first evaluation of the protocols when they are subjected to DoS attacks is to monitor the behaviour of

a SPAN Responder and an IKEv2 Responder (with COOKIE being used) under different types of DoS attack messages. For each DoS attack evaluation, 100 trails of the Responder to detect DoS attacks were monitored. The average time delay is the (average) time needed by a Responder to detect a DoS attack. This time delay enables one to determine the reply time of the Responder when it is subjected to DoS attacks. Thus, the Responder that is quicker to detecting DoS attacks may terminate DoS attacks quicker, and therefore fewer resources would be wasted, and therefore it is more resistant to DoS attacks.

In the first evaluation, the attacker is configured to send to an IKEv2 Responder legitimate IKEv2 message 1 and 3 (with valid COOKIE), and it signs IKEv2 message 5 with an *illegitimate* public key pair. On the other hand, the attacker sends an illegitimate SPAN_INIT message (with invalid signature) to a SPAN Responder. The average DoS attack detection time of the SPAN Responder and an IKEv2 Responder are 235 milliseconds (±57ms) and 2402 milliseconds (±354ms) respectively (~90% less). This result is in-line with the previous discussion: that SPAN detects DoS attacks much quicker than IKEv2. This is because under this form of DoS attack, an IKEv2 Responder would be wasting resources on creating COOKIE (upon receiving IKEv2 message 1), computing new D-H exponentials (upon receiving IKEv2 message 3), computing shared key set (upon receiving IKEv2 message 5), decrypting the encrypted payload in IKEv2 message 5, and checking the (invalid) digital signature in IKEv2 message 5.

To accommodate the performance costly D-H exponential computation process, the IKEv2 RFC recommends several ways to *reuse* D-H exponentials at the expense of having less-than-perfect forward secrecy, or maintaining more state.

Thus, another evaluation is carried out, in which the process is identical to the previous experiment except that D-H exponentials are reused in IKEv2. The results show that on average it takes 883 milliseconds (±347ms) for an IKEv2 Responder to detect a DoS attack i.e. SPAN has ~73% less in time delay.

Although the time taken to carry out an operation does not necessary give any direct indication on the actual resource needed to carry out the operation (i.e. CPU power, memory used), this thesis argues that it is a good indication, that is related to the consumed amount of resources for carrying out the operation (i.e. more time required means it is likely that more resources are needed). One can imagine that if each Responder is implemented as multiple threads, and each thread handles each call (initiated from the Initiator), the less time the thread needs to detect a DoS attack, the better the design. This is because the more threads that are alive at one time, the more resources are consumed at one time for the node to manage each thread (and too many threads at one time may eventually overload the node, which is what DoS attacks are for). Therefore, by comparing the time needed to detect DoS attacks between different approaches, this gives the readers an insight on the differences between the behaviours of the approaches under DoS attacks.

5.6 Evaluation on Robustness & Flexibility

Evaluation on the robustness and flexibility of the approaches are based on how the protocols would operate in a *heterogeneous* networking environment in two aspects: a) verification of static code execution compatibility, and b) support for security parameter negotiations.

SPAN has the [EEi] field as an optional field to accommodate code execution incompatibility. By having this field in the first message, the Initiator may

propagate authenticity and integrity protected code execution requirements to potential Responder(s), prior to any further processing. Potential Responders can determine compatibility prior to continuing any further processing. Potential Responders must evaluate their own systems, to ensure that they can execute the to-be-sent active packet (i.e. either their systems fully satisfy the Initiator's requirements, or as long as the Initiator makes some accommodations on the static code). Therefore, prior to establishing hop-by-hop SA and subsequently active packet transmission, the Initiator could be certain that the Responder is capable of executing the active packet. In other words, the Initiator will not be wasting resources on establishing hop-by-hop SA with incompatible Responder. In this way, SPAN improves robustness of the underlying active networking systems, through ensuring compatibility of code execution on remote nodes prior to hop-by-hop SA establishment and active packet transmission.

SPAN enhances flexibility by enabling hop-by-hop SA negotiations between nodes. By using the SAi and SAr fields, nodes can negotiate security parameters e.g. supported/preferred encryption algorithms and key size in a hop-by-hop manner. It should be noted SANTS does not address hop-by-hop key establishment; whereas pre-distributed shared key, SKT, and SANE do not support SA negotiation. As such, these approaches have limited flexibility when deployed in a heterogeneous environment, because individual security needs may not be satisfied.

6 Conclusions

The thesis began with an introduction to the fundamental concepts of active networking and one of its major security challenges, namely, hop-by-hop security. Existing hop-by-hop solutions for active networks were discussed. The author's solution, i.e. SPAN, was presented, discussed, and evaluated.

It was discussed in section 1.5 (p.20) that an active network consists of a mixture of active nodes and passive nodes in the Internet. An active node is a passive node equipped with an active platform. The NodeOS of an active platform hosts several functions to serve specific AAs or EEs' needs. Example functions are security, packet de-multiplexing, resource control... and more.

An active node is capable of intercepting active packets, and executing the code carried in the packets. Active packets carrying active code are created by a principal (e.g. a management application), and are injected into the active network at the source node. Active packets are different from passive packets, that active packets may carry static executable code and dynamic execution results. At each intercepting node, the packet's static code is executed. The execution results may be added back to the packet before the packet is forwarded to its next hop. This means that the contents of active packets are subjected to modifications whilst the packets are traversing the network. This thesis regarded this feature as the dynamic nature of active packets.

In section 1.6 (p.28), it was discussed that active packets must support dynamic routing (i.e. pre-specified route of active packet is neither scalable nor practical). This is because to enforce static routing, the principal/source node must have in-depth knowledge of the entire network, and must assume stable network conditions. Dynamic routing means the next hop of an active packet is

determined at the node of execution based on execution results and real-time network conditions. Dynamic routing is therefore a key feature to enhance flexibility in active networking technologies.

Also in section 1.6 (p.28), it was further discussed that there is a need for a hop-by-hop security approach for heterogeneous large-scale active networking systems. The need was due to the dynamic nature and dynamic routing capability of active packets. A hop-by-hop security approach means active packets are protected in a hop-by-hop manner, so that modifications on an active packet made by an intermediate intercepting node are verifiable: the authenticity and integrity of the dynamic data on active packets should be verified by the identity of the node that the packets were last modified. Furthermore, this thesis has also identified that the static code of an active packet must be protected, but based on the identity of the principal. This is because the static code is created by the principal, and the same piece of code is expected to be executed on all nodes without subjected to changes. Therefore, the authenticity and integrity of static code should be verified against its actual creator i.e. the principal. Static code should also be subjected to non-repudiation protection. This is because static code may be created for control purposes on remote nodes, and compromised control operations may lead to potential damage on remote nodes. Therefore, static code should be protected in a way that its creator cannot deny any wrongdoing. Lastly, confidentiality protection should be enforced on active packets, so that attackers cannot determine their contents.

Existing solutions to the problem space were discussed in chapter 2 (p.37). The drawbacks of related work in terms of scalability, efficiency, flexibility,

robustness and security were discussed. Related work either does not support SA negotiation, which limits its flexibility to cope with heterogeneity between nodes of different administrative domains (e.g. pre-distributed shared key, SKT, SANE). Related work follows a centralised approach which is not scalable (e.g. KSV), or does not provision for key management (e.g. SANTS), or without arrangements to optimise performance for hop-by-hop deployment (e.g. IKEv2); or provides no identity protection to both the Initiator and the Responder (e.g. IKEv1 in aggressive mode+IPSec and JFK); or wastes more resource prior to detecting DoS attacks (e.g. IKEv1 in aggressive mode, IKEv2 and JFK). Existing solutions do not improve robustness of the underlying active network system. Pre-distributing shared keys to all nodes in the network is neither scalable nor practical; asymmetric operations incur a relatively much higher overhead that makes it much less efficient, and would not scale well when the number of active packets to be protected is large. Therefore, this thesis investigated a new hop-by-hop SA establishment technique to address these problems.

To accommodate these new hop-by-hop security challenges in active networks, this thesis presented SPAN in chapter 3 (p.74). SPAN is a *secure*, *scalable*, *efficient*, and *flexible* hop-by-hop security approach for large-scale active networking systems, that enables secured EE information exchange prior to actual hop-by-hop SA establishment, and enables active packet transmission *during* hop-by-hop SA negotiation, instead of after. SPAN is designed against replay, man-in-the-middle, impersonation attacks, and is capable of determining DoS attacks much more quickly. In SPAN, three messages (SPAN_INIT, SPAN_AUTH, and SPAN_AP) are exchanged to complete one hop-by-hop SA

establishment *and* secure active packet transmission. The first two messages enable the Initiator and the Responder to determine compatibility for active packet execution, to verify each other's authenticity, to negotiate SA parameters, and to establish a shared key set. The last message enables the Initiator to complete the protocol, and to securely transmit an active packet to the Responder.

In Chapter 4 (p.98), the features and advantages of SPAN were discussed, with comparisons to existing solutions. Unlike existing approaches, SPAN is more efficient because it has a better design that reduces the number of message processing and cryptographic operations that are required to complete a protocol exchange. The provisioning in SPAN for making remote EE queries prior to hop-by-hop SA establishment and actual active packet transmission enhances the overall robustness and efficiency of the underlying active networking systems. This is because incompatible Responder will be automatically excluded from the hop-by-hop SA establishment process, and therefore no resources will be wasted by the Initiator on establishing hop-by-hop SA with incompatible Responders. SPAN is fully distributed, in the sense that every node is capable of negotiating and establishing hop-by-hop SA with its (overlay) neighbours. SPAN is scalable, that requires no centralised server, pre-established trust, or feedback system, and requires the minimum number of message exchanges to complete the protocol. SPAN is flexible in the sense that it provisions for hop-by-hop SA negotiations between nodes, so that tailored made hop-by-hop SAs can be established.

Also in chapter 4 (p.98), it was further discussed that the authenticity, integrity and confidentially of active packets are protected in SPAN. SPAN is designed

to be resistant to replay, man-in-the-middle, impersonate attacks. This is because the design of SPAN includes appropriate defence mechanisms such as nonces, countersigned nonces, symmetric cryptographic protection, and asymmetric cryptographic protection. Furthermore, SPAN requires each peer to verify the shared key set as soon as the key set is computed. This process enables the peers to discover any computational errors (of computing the shared key set) at the earliest stage of the protocol as possible. Applications' identities (EEs/AAs) are not revealed to the peer unless the authenticity of the peer is verified. SPAN is designed to detect DoS attack much more rapidly than existing approaches. This is because SPAN restricts all computation processing until the peer's authenticity has been verified. Provisioning was made in the design of SPAN to limit the use of performance-costly asymmetric cryptography to strong and essential security only. Provisioning has also been made to accommodate scalability issues of using asymmetric cryptography in a large-scale network.

In the evaluation chapter (chapter 5 on p.121), the prototypes for SPAN and related solution(s) (i.e. variants of IKEv2) that were developed for evaluation purposes were presented, together with their implementations. Promising results of SPAN were then presented and discussed. SPAN achieves on average 15% to 40% less in performance overhead when compared to some of related approaches; and SPAN is designed to detect DoS attacks much more efficiently than some existing approaches (i.e. 73 to 90% less in time delay for detection time for DoS attacks). The high level of robustness, flexibility, and scalability of SPAN was also presented.

6.1 Applying SPAN to Other Areas

SPAN was designed for hop-by-hop security in active networks. However, potentially, certain features of SPAN could be re-applied to networking environments other than active networks, in particular in areas where hop-by-hop protection is required.

One potential area where hop-by-hop protection would be needed is distributed management systems. The author realised this opportunity whilst he was developing different distributed systems [85][86][87][88] in recent years. The author has identified several similarities between distributed management systems and active networking systems:

- Both types of system are distributed (i.e. each participating active nodes may distribute active packets); so as each participating distributed management nodes may distribute management instructions.
- In both types of system, active packets/management instructions are to be executed on (a set of) local and/or remote nodes.
- Scalability is a critical design factor in both types of system i.e. must support large-scale deployment.

Although there are similarities between the two types of system, this does not necessary mean the *same* hop-by-hop security protocol, i.e. SPAN, can be applied directly to secure management instructions in distributed management systems, without modifications. This is because, to the best of the author's knowledge, there is little evidence to suggest that distributed management packets have dynamic nature. More specifically, in [89][90], distributed management packets (known as explorer packets) are propagated from one source to the entire Internet. Each intercepting node of these explorer packets

takes note of the information contained in the packets, and creates a parent-child relationship with the node from which the packet was previously delivered. This parent-child relationship forms a spanning tree in the network. When the explorer packets reach the edge of the network, they bounce back as echo packets to the source, by following the spanning tree that was established when the explorer packet was propagated through the network.

The similarity between active networks and the distributed management systems is that the echo packets contain some executable code. Each node intercepts the echo packets, and executes the code. It is foreseeable and apparent that the echo packets may also carry the intermediate execution results (such as the IP addresses of traversed nodes). Given that there are some similarities between distributed management systems and active networking systems, the author argues that, the research work conducted and presented in this thesis would enable the readers to gain a better understanding of the distributed nature of active/distributed management systems, and the challenges and design requirements of hop-by-hop security. These experiences may put the readers in a better position to evaluate, experiment with, and to investigate the possibility of (re)developing SPAN into different models for different distributed management systems.

7 Publication List

- Below is a selected list of papers published by the author.

1. L. Cheng, K. Jean, R. Ocampo, A. Galis, P. Kersch, R. Szabo, "Secure Bootstrapping of Distributed Hash Tables in Dynamic Wireless Networks", in Proceedings of International Conference on Communications (ICC), Glasgow, Scotland, UK, June 2007.

2. L. Cheng, A. Galis, "Security Protocol for Active Networks", to appear in Proceedings of the 14th IEEE International Conference on Networks (ICON), Singapore, Sep 2006.

3. L. Cheng, A. Galis, "Simple Key Exchange for Active Networks", in Proceedings of the 13th IEEE International Conference on Networks (ICON), Kuala Lumpur, Malaysia, Nov 2005, Vol. 1, pp. 6-11.

4. L. Cheng, A. Galis, C. Todd, "Active Network Authentication", in Proceedings of London Communication Symposium (LCS), London, UK, Sep 2003.

8 Appendix

8.1 Certificates & Public Key Infrastructure (PKI)

Public keys are asymmetric cryptography (section 8.5.2 on p.165). Public keys are commonly used as encryption keys and authentication keys during communication processes. However, the "man in the middle" attack (section 8.5.4 on p.167) shows that the use of public key on its own (i.e. without further protection) may lead to integrity attacks. This is because without additional security precautions, the authenticity of public keys may be forged. Therefore, authenticity of public keys must be verified before the keys can be used. Digital certificates are introduced to prevent public keys forgery: they are known as public key certificates. These certificates use digital signature to bind together a legitimate public key with its owner's identity. The signature is created by a reliable Certificate Authority (CA) (section 8.1.1 on p.162). The entire system of public key distribution, certificate creation and verification is known as the Public Key Infrastructure (PKI) [91].

8.1.1 Certificate Creation

The fundamental requirement of creating a certificate is that the end user/client must have a public key pair. Web browsers are capable of generating public key pairs and so as some other software such as keytool provided with Java SDK [92][105]. Once a public key pair is created, the client (on behalf of its end user) sends the following to a reliable CA for certificate creation:

- A certificate request
- The end user's name
- The end user's public key

Verisign [82] is an example CA. The client should keep the end user's private key locally and securely. Upon receiving the name of the end user and the public key of the end user from a client, the CA verifies whether the name claimed to be the end user really belongs to the end user. This can be done through checking business registration records (for companies) or passport/driving licenses (for individuals). This is an administrative issue and therefore shall not be discussed in this thesis.

Once the received information is authenticated, the CA creates a message (m) based on the materials received from the end user. This message is then signed by the CA using the CA's private key. The resultant signature (s) is sent along with message m to the end user/client. Note that message m contains the end user's public key. Message m and signature s together become the end user's certificate [93].

A X.509 certificate contains essential information about the end user (i.e. his/her name and his/her public key, and optionally additional information of the end user e.g. e-mail addresses). Thus, certificates are not restricted for public key verification only. A copy of the certificate may be distributed (if necessary) or uploaded and published in the CA's directory. A CA manages a Certificate Server that manages certificate storage and publication.

A certificate would become invalid once it has expired. A CA can actively expire a certificate by listing the certificate in the Certificate Revocation List (CRL), which is distributed publicly.

8.1.2 Certificate Verification

The authenticity of a certificate (hence the authenticity of the embedded public key on the certificate) is verified by checking the digital signature on the

certificate. Web browsers contain a list of CAs' public keys. These public keys are used by the Web browsers to authenticate the digital signatures on certificates. Note that the digital signatures on the certificates are created by the CA's private key. The matching public key will be selected by the Web browser for digital signature verification. Once the digital signature is verified, subsequently, the embedded public key on the certificate is verified. The public key is said to be authenticated. Thus, the "Man in the Middle" attack described in section 8.5.4 (p.167) would not succeed.

8.2 Concatenation

The notation for concatenation is |. Concatenation puts two strings together, one appending the other. For instance, concatenating two strings "HELLO" and "WORLD" gives "HELLOWORLD".

8.3 Cookies

The term cookies originated from [94]. Cookies were used in IKEv1 as SA identifiers. Thus, cookies are equivalent to *SPI* in IKEv2 (section 8.17.1 on p.192). In IKEv1, there are two cookies in each ISAKMP message, one cookie is the Initiator's cookie, the other one is the Responder's cookie. The cookies refer to the Initiator's IKE SA and the Responder's IKE SA respectively. Each cookie is eight *octets* i.e. 64-bit.

8.4 Credentials

Credentials are used in authentication and access control. They bind an object of identity to a claimant's property such as IP address. A verifier must verify the credentials during an authentication process. A verified credential unique identifies the claimant. For example, a digital certificate is an electronic

credential that binds the identity of a public key owner to his/her public key.

8.5 Cryptography

8.5.1 Symmetric Cryptography

There are two types of cryptographic techniques:

- Symmetric
- Asymmetric

Symmetric cryptography is commonly used as an encryption technique for protecting message confidentiality. It differs from asymmetric cryptography in the sense that it uses the same key for encryption and decryption; whereas asymmetric cryptography uses different keys for encryption and decryption respectively. In symmetric cryptography, the key is shared between the message sender and the receiver. Symmetric cryptography may also be used for hashing, integrity checks and authentication.

8.5.2 Asymmetric Cryptography

Asymmetric cryptography makes use of a public key pair for authentication and integrity protection. It may also be used for confidentiality protection. Assume a receiver needs to verify the authenticity of a block of data that was digitally sign by another entity (known as the signer in this thesis). The signer must first generate a public-private key pair (section 8.1 on p.162). Note that this pair of keys must be unique i.e. only one can verify the data signed by the other. The private key must be securely kept by the signer, whereas the public key will be distributed to other parties in the Internet. Public keys are usually distributed in the form of certificates (section 8.1 on p.162). The corresponding public key certificates must be accessible by the receiver. The receiver will then obtain a public key of the signer. The signer then signs the data with its private key, and

sends the signed block of data for verification.

For confidentiality protection, the process is similar except that the signer encrypts the data using the receiver's public key. Since only the receiver owns the corresponding private key, the confidentiality of the data is protected. However, authenticity is not protected. These issues are addressed in section 2.6 (p.48).

8.5.3 Symmetric vs. Asymmetric

Symmetric operations are said to be much faster than asymmetric cryptography operations [95]. Thus, symmetric cryptography is used for protecting large chunk of data, whereas asymmetric cryptography is used for protecting small size data, such as message digest.

Question 6: Explain the reasons why symmetric operations are faster than asymmetric operations.

Asymmetric operations (i.e. RSA) generate ciphertext as follow:

$$C = P^e \bmod N$$

Equation 11

C is the ciphertext, P is the plaintext (the text to be signed), e and N are values chosen and used by the RSA algorithm.

Equation 11 essentially means: C is the remainder of P^e/N. Note that P must be exponentially multiplied by e. Therefore, if e is large, it becomes computationally expensive to calculate P^e. However, e must be large for strong *encryption* security [96].

On the other hand, although different symmetric techniques use different ways to encrypt (scramble) plaintext, but symmetric operations use XOR operations. XOR operations are more difficult to implement in software, and are usually

implemented in hardware instead. This is because XOR involves bit relocation, and bit re-location can be easily performed with wired hardware. However, because XOR operations involve only bit re-location, they are faster than exponential multiplication. Therefore, symmetric operations are faster than asymmetric operations.

Symmetric cryptography requires secret key establishment between two or more participating clients across an insecure link (e.g. Diffie-Hellman key exchange between two hosts over the Internet), thus symmetric keys distribution must be authenticated, integrity protected and confidentiality protected. Strong security requirements in symmetric key distribution create a key distribution challenge. Asymmetric cryptography does not require secrete key distribution, only the public key distribution is needed. Because public keys are meant to be public, thus public keys do not require confidentiality protection. Thus key distribution is less challenging in asymmetric cryptography (only authentication and integrity protection are required). Asymmetric keys are usually distributed in the form of digital certificate. A well-known support architecture for certification verification is the PKI (section 8.1 on p.162). Without support from PKI, public keys are subjected to Man-in-the-Middle attacks (section 8.5.4 on p.167).

8.5.4 The "Man in the Middle" Attack of Public Key Cryptography

This example illustrates how the authenticity of public keys may be forged between an end user and a serve in the absence of certificate. Image a client (i.e. a web browser) is requesting on behalf of its end user for a secure connection to an on-line bank server. An attacker is in between the client and the server, and he is capable of intercepting the traffic between the client and

the server.

If the on-line bank server replies to the client with its public key only (i.e. no certificate):

1. The attacker intercepts the on-line bank's public key.
2. The attacker sends his own public key to the client, pretending that the key belongs to the on-line bank's server.
3. The client receives the (forged) key, and thinks the key really belongs to the bank.
4. The client encrypts an instruction (e.g. "pay Mr Smith 60 pounds now") with the (forged) key, and sends the encrypted instruction to the bank. The client expects that only the bank (which owns the corresponding private key) may decrypt the encrypted instruction.
5. The attacker intercepts the encrypted instruction, and obtains the end user's instructions by decrypting the encrypted instruction using his private key.
6. The attacker modifies the end user's instruction with "paid the attacker 60 pounds now" and encrypts the new instruction by the bank's public key, then sends the encrypted instruction to the bank. In this case, neither the end user nor the bank would recognise the existence of the attacker until the transaction raises a concern.

This problem can be resolved by authenticating public keys through certificates (see section 8.1 on p.162).

8.6 Diffie-Hellman Key Exchange (D-H)

Symmetric cryptography needs secure symmetric key sharing between participating parties over an insecure channel. Diffie-Hellman (D-H) key

exchange is a secure method for establishing symmetric keys between parties over an insecure channel. Note that this method is used to protect the confidentiality of the shared key. It does not protect the integrity or authenticity of the exchanged keys. It should be note that D-H is the underlying *theory* of key exchange. Internet Key Exchange (IKE) (section 2.10 on p.55) uses D-H to establish shared secrets, and IKE provides authenticity and integrity protection to the exchanged shared secret.

8.6.1 Diffie-Hellman Key Exchange in MODP Mode

In general, mathematical functions are two-way functions, for example:

$$y = e^x$$

Equation 12

Rearrange Equation 12, the value of x can be determined by:

$$x = \log_{10} y$$

Equation 13

However, the D-H algorithm uses a *one-way* function:

$$f(x) = x \bmod p$$

Equation 14

Since the function in Equation 14 is a one-way function, the value of x in Equation 14 cannot be determined through reverse engineering of the function. Basically, Equation 14 means f(x) is the remainder of x/p.

The steps of the D-H algorithm in MODP mode are:

1) The participating parties e.g. Client A and Client B pre-agree on two values: L and P.

Rule 1: L and P are values that are needed for Equation 15 (see later). L must be < P (see later).

2) The participating parties select their own private secret: Client A: a, Client B: b.

3) Each party uses its own private secret and the one-way function (Equation 14) to generate a *public sharable value* respectively: α and ß.

Client A: $\alpha = L^a \bmod P$

Client B: $\beta = L^b \bmod P$

Equation 15

4) The parties exchange the public sharable value. For example, client A sends α to client B, and client B sends ß to client A.

5) Each party uses its own private secret, the public sharable value (that was received from the other party), and the one-way function to generate a symmetric shared secret.

Client A's symmetric shared secret = $\beta^a \bmod P$

Client B's symmetric shared secret = $\alpha^b \bmod P$

Equation 16

Note that the resultant secret is *symmetric* i.e. Client A's and Client B's shared secrets are identical and can be used for encryption and decryption.

Example 1: Given that L = 5, P = 13, a = 4, b = 7. Determine the symmetric shared secret [40].

First, determine the public sharable value α of client A. According to Equation 15:

$\alpha = L^a \bmod P$

$\alpha = 5^4 \bmod 13$

$\alpha = 624 \bmod 13 = 1$

Using the same process:

ß = 8

Once α and ß are exchanged between Client A and Client B, determine Client A's symmetric shared secret:

$= \beta^a \bmod P$

$= 8^4 \bmod 13$

$= 4096 \bmod 13$

= 1

Then determine Client B's symmetric shared secret:

$= \alpha^b \bmod P$

$= 1^7 \bmod 13$

= 1

Therefore, the symmetric shared secret is 1. Client A and Client B owns an identical shared secret.

Since at each party the public sharable value is generated by using the party's own private secret, the public value is therefore related to the private secret. Since the public shareable value is generated by using the one-way function, the private secret used to generate the public shareable value cannot be determined from the public shareable value. This is why it is safe to transmit public sharable values across an insecure channel.

Question 7: Explain why the nodes end up with the identical symmetric shared

secret in the Diffie-Hellman protocol, given that no secrets are exchanged.

The reason why the symmetric shared secret are identical is because both secrets are generated using the same ingredients (i.e. L, P, α, β). Extending Equation 16 (p.170):

Client A's symmetric shared secret:

$= \beta^a \bmod P$

Since $\beta = L^b \bmod P$, therefore Client A's symmetric shared secret:

$= [L^b \bmod P]^a \bmod P$

$= L^{ab} \bmod P$

Similarly, Client B's symmetric shared secret:

$= \alpha^b \bmod P$

Since $\alpha = L^a \bmod P$, therefore:

$= [L^a \bmod P]^b \bmod P$

$= L^{ba} \bmod P$ which is the same as Client A's symmetric shared secret

8.6.2 Selection of Private Values

There are certain rules when selected the private values (a, b) and the prime modulo P. If these values are small (in particularly if P is small), D-H becomes insecure.

This is because the shared secret value = $L^{ab} \bmod P$. This means that the shared secret value is the *remainder* of L^{ab}/P, which means the shared secret value could be any value between 1 to P-1 only. Thus, if P is small, an attacker may easily guess what the shared secret value is. P should be selected a prime

number larger than 300 bits, whereas a and b should be at least 100 bits. According to [97], P should be between 512-bit to 1024-bit, whereas a and b should be an integer between 1 < a < P-2. a and b should be 192-bit and an integer value less than (P-1)/2.

8.6.3 Selection of the public values

The public values L and P are carefully selected: L < P, and L must be a primitive root for P if the powers of L, L^0, L^1, L^2, L^3, ... include all the residue classes mod P (except 0). The primitive roots of P are the appropriate value(s) for L (that are between 1 and P-1) – when used in the mod operation i.e. L^e mod P (where e is any exponential) – that return the remainders that cover every number mod P occurs except 0. More specifically, the function should return the remainders between 1 to P-1 for different values of e. To illustrate this point, an example is given below:

Example 2: Determine the appropriate values for L when P = 7.

Consider this equation L^e mod P, the following table can be constructed. The first column contains different values for e, the first row contains different values for L. The table contains the remainders for different combinations of L and e. Note that the value for P is fixed (i.e. P = 7).

	Different values for L					
e	1	2	3	4	5	6
0	1	1	1	1	1	1
1	1	2	3	4	5	6
2	1	4	2	2	4	1
3	1	1	6	1	6	6

4	1	2	4	4	2	1
5	1	4	5	2	3	6
6	1	1	1	1	1	1
7	1	2	3	4	5	6
9	1	4	2	2	4	1

Table 1 – The remainders for L^e mod P

For example, if e = 0 and L = 1, the remainder = 1^0 mod 7 = 1. If e = 3 and L = 4, the remainder = 4^3 mod 7 = 1. Note that in this example (i.e. when P = 7), L = 3 or L = 5, and L < P.

The reason why L should be 3 or 5 is that only when L = 3 or L = 5, the remainders cover all the values between 1 to P-1, that is, between 1 to 6. Thus, 3 and 5 are the primitive roots for P when P = 7.

8.6.4 Limitations of the Diffie-Hellman Algorithm

The process requires the participating parties to be on-line *at the same time* during the key exchange process. D-H can only be deployed when both parties are on-line at the same time. This is because the parties must exchange certain values during the key exchange process. This is a not a crucial limitation because both parties must be on-line anyway if they want to exchange keys. One party cannot exchange key with another party if the originator cannot be in contact with the other party. If off-line key exchange is required, one may suggest caching the exchange values. However, caching is not a scalable solution because the proxy would have to keep track of which public value belongs to which party at what time and so on.

No authentication and integrity protection: a man-in-the-middle attack could be easily performed. The lack of authentication and integrity protection is a more

severe limitation. However, this is an application issue and does not devalue the *theory* itself. D-H itself is just a theory and it is commonly used for secret key exchange because it is proven to work and its simplicity. As a result, the D-H algorithm is used with additional precautions. For example, the D-H algorithm is used with additional security measurements in IKEv2 (section 2.10 on p.55).

8.7 Hash, Keyed Hash, Hash Functions, Hash Tables

Asymmetric cryptography subjects to two drawbacks: performance and plaintext size restriction. Digital signature computation is known for its slow performance (Question 6 on p.166). Also, the size of the plaintext to be encrypted is limited. To overcome these drawbacks, the digital signature is made by signing a *hash* of the plaintext, instead of the plaintext itself.

A hash is also known as a *Message Digest* (MD) or checksum. A hash is computed by applying a *hash function* on a piece plaintext. The resultant hash is much smaller than the size of the plaintext. The size of a hash/MD/checksum is usually fixed, whereas the size of the plaintext is arbitrary.

Apart for being used as a form of compression technique, hashes are generally used as checksums (i.e. for integrity checks on exchanged messages between network entities). The message sender creates a hash of the message to be exchanged, and sends the hash along with the message to the receiver. Upon receiving the message at the message receiver, the receiver calculates a hash based on the message received, and compare the two hashes as part of the integrity check process.

Authenticity protection on the hashed data is achieved when a key is used during the hashing process. The result is known as *keyed hash*. One example is Hashed Message Authentication Code (HMAC) [45], which is used in IPSec

for integrity protection and authentication.

A hash function is one-way: the resultant hash cannot be used to reproduce the plaintext. A secure hash function would only generate a particular hash from a particular piece of plaintext, no two (different) plaintext would result in the same hash. If two different pieces of plaintext result in the same hash, then a *collision* is found. There are several types of hash functions [98].

8.7.1 Keyed Hash Functions

Keyed hashes are used for authenticity and integrity protection. To use keyed hash functions, the message sender must share a key with the message receiver in advance. The message sender uses the shared key and a particular hash function to generate a fixed-length hash of a variable length message to be transmitted. Upon receiving the message and the fixed-length hash at the receiver, the receiver computes a hash of the received message by using the same shared key and the same hash function and compares the two hashes. If the two hashes match, the integrity of the message is verified.

8.7.2 Message Authentication Code (MAC)

Message Authentication Code (MAC) is keyed hash function that uses shared symmetric key cryptography to protect message integrity and authenticity. The shared key is hashed with the message to be protected to generate a MAC. The resultant MAC is appended to the message to be checked by the recipient.

Note that MAC provides integrity protection as well as authentication (only the other participant that owns the shared symmetric key could have generated the MAC). However, MAC should not be considered as digital signature because MAC does not provide non-repudiation. Digitally signing a piece of data provides both authentication and non-repudiation protection to the signed data

because only the signer has the private key to create the signature.

8.7.3 Hashed Message Authentication Code (HMAC)

HMAC is a special type of keyed hash. It works with other existing hash functions such as SHA and MD5 i.e. HMAC-SHA and HMAC-MD5. The fundamental principle of HMAC is to generate a keyed hash of a keyed hash, thus it is stronger. HMAC is designed to provide additional security to existing hash functions, without making any modification to the hashing algorithms themselves. The following shows the fundamental concepts of HMAC:

$$HMAC(K, P) = H[(K \text{ XOR opad}) \mid H(K \text{ XOR ipad} \mid P)]$$

Where
H = a selected underlying cryptographic hash function e.g. MD5, SHA.
K = the length of the shared symmetric key (should be > L but □ B)
P = plaintext
opad = an outer pad (a fixed length string)
ipad = an inner pad (a fixed length string)
B = hash function block length = 64 bytes for MD5 and SHA-1
L = hash function output length = 16 bytes for MD5, 20 bytes for SHA-1

Equation 17

8.8 Initialisation Vector (IV)

When using the same key for encryption repeatedly, a security weakness is found in block and stream ciphers. When encrypting with block ciphers in CBC mode, a (large) message is split into a series of small blocks prior to encryption. To encrypt the message, each block of the message is XOR-ed with the previous (encrypted) block. However, if the same key is used, two similar messages would end up with two similar ciphertext (except the blocks which in the ciphertext that contain the differences).

Stream ciphers are subjected to this weakness as well. If a stream cipher is used with the same key, XOR-ing two pieces of ciphertext would result in a

XOR-ed version of the two plaintext that were encrypted. If the plaintext were written in human readable format then information would be discovered by the attacker.

However, key generation processes are performance-wise expensive. It is not scalable and practical to request all security systems to generate (and share) a new symmetric key every time prior to encryption. The problem can be solved by adding randomness to the encryption process. The plaintext is pre-pended with a randomly selected IV block. Thus, even two similar pieces of plaintext are to be encrypted in CBC mode or stream ciphers, the actual message to be encrypted are different because different randomly selected IVs are added to the plaintext respectively. Thus, the resultant ciphertext would be unique.

The length of IV must be the same as the block size of the cipher. This is because when the cipher is operated in CBC mode, the first block of the plaintext is XOR-ed with the IV. The IV must be sent along with the ciphertext to the recipient so that the recipient knows what value of IV to use for decryption.

8.9 Initiator

An initiator is an entity that instantiates a *Security Association* (SA) (section 8.16 on p.189) negotiation. A SA negotiation involves at least two entities: the Initiator and the *Responder* (section 8.15 on p.189).

8.10 Nonces

Replay attack is an attack of which an attacker copies a (valid) message, and re-uses the message at a later stage to fool the receiver. Nonces are included in messages for anti-replay attacks (section 8.13.2 on p.183). Alternatively, sequence numbers may be used for anti-replay attacks (section 8.18 on p.194). For instance, nonces are used in IKEv2 [30].

The idea of nonces is to add some randomness to the message. For instance, in IKEv2, the IKE_SA_INIT message contains a randomly generated nonce that is at least 128-bit and at least half of the PRF function that is used for shared key generation (section 8.13 on p.182). Both the Initiator and the Responder keeps a record of the nonces that are used and received. Nonces should never be re-used during a session. If the Responder receives two messages of the same nonce, the Responder will consider the second message as a replay attack.

Thus, nonces add freshness to the messages. In some cases, nonces add freshness to keys too. For instance, in IKEv2, nonces are used for shared key generation (section 2.10 on p.55)

8.11 Passive Network

A passive network composes of *passive nodes* only. It is a *store-and-forward* network in which *passive packets* are forwarded to their desired destination by passive nodes. The content carried in passive packets is irrelevant to passive nodes in a passive network.

- Passive Node

A passive node is a router or a switch that performs simple packet forwarding function. There are hardware routers (i.e. high-speed IP routers used for packet routing in the Internet) and software routers (i.e. Linux PCs with multiple Ethernet cards and with ip_forwarding enabled).

- Passive Packet

These are normal IPv4 (or IPv6) packets routed by passive nodes in a passive network.

8.12 Perfect Forward Secrecy (PFS)

8.12.1 Definitions

Perfect Forward Secrecy (PFS) refers to a property of key generation processes. PFS was first defined in [79]: a key generation process is said to support PFS, if the secrecy of generated keys would not be compromised even if the long-term secret key (that was used to generate subsequent keys) were disclosed. In other words, with PFS, if a root key (that was used to generate subsequent keys) is compromised, subsequent keys should not be compromised.

In some other documents [30], PFS has a slightly different definition. PFS may be defined as: once a connection is closed and its corresponding keys are forgotten, even someone who has recorded all of the data from the connection and gets access to all of the long-term keys of the two endpoints would not be able to reconstruct the keys used to protect the conversation without doing a brute force search of the session key space.

The major difference between the two definitions is that the latter refers to a situation when a connection is closed; whereas the former requires PFS in any situation. The former definition implies that, when generating subsequent shared keys by using a pre-established key, the key generation parameters that were used for generating the pre-established key should *not* be re-used for the subsequent keys generation. It is obvious that if the key generation parameters are re-used for generating subsequent keys, and if the key generation parameters are compromised, then *all* subsequent keys would also be compromised (i.e. a chain effect). A chain of keys is a bad design.

However, if new key generation parameters were generated for each key

establishment process, the performance overhead would be enormous. For instance, a common key generation parameter is the Diffie-Hellman parameters (section 8.6 on p.168); and D-H parameters generation is known to be expensive process in terms of performance due to the exponential and modular calculations involved, essentially the same problem as asymmetric cryptographic techniques (section 8.5.3 on p.166). Thus, some cryptographic designers suggest another definition of PFS (i.e. the second definition), which in their point-of-view, is sufficient.

The second definition requires all nodes to forget the key generation materials once the connection has closed. This implies that, under this revised definition, key generation materials may be re-used during a specific period (i.e. when the connection is still on). This period refers to the period of which the pre-established keys are still valid. This definition is made on the assumption that the effective period of the pre-established keys are short (e.g. 8 hours), therefore, it is unlikely that – within that short period - an attacker would have hacked the pre-established keys, and hence compromising the subsequent keys. Some serious cryptography designers, however, believe that this arrangement is too optimistic, and so call less-than-perfect forward secrecy. Never the less, the second definition leads to a less than ideal design (i.e. a chain of keys), but is more performance-wise practical because of the performance overhead saved.

There is an on-going debate on which definition should be adopted. This thesis refers to the original definition as defined in [79]. Firstly, this definition is original; secondly, this definition results in a better design; thirdly, the requirement of this definition is higher (that each key generation process requires new key

generation parameters), and is therefore more challenging. The author believes that, rather than using a simplified definition; effort should be made to targeting the requirement that is more complicated.

8.12.2 PFS Explained

In IKEv2, a root secret (i.e. SKEYSEED) is generated during the IKE_SA negotiation. If the same root secret is (re)used to derive subsequent keys during the CHILD_SA negotiation (for example, IPSec SA negotiation), then the system does *not* support PFS. To enforce PFS, new Diffie-Hellman values should be used for the CHILD_SA negotiation.

Note that PFS is needed for strong security. Without PFS, if an attacker obtained the root secret, he would be able to determine the subsequent keys. Recalling from section 2.10.2 (p.57), subsequent keys (such as SK_a, SK_e) are generated by using publicly accessible data such as $NONCE_i$, $NONCE_r$, and SK_d. If SKEYSEED were compromised, then SK_d would be compromised. Hence, all subsequent keys would be compromised as well.

However, Diffie-Hellman values generation is computationally expensive. Thus, PFS is only enforced for security applications that require high level of security. In IKEv2, including new Diffie-Hellman values during CHILD_SA negotiation is optional.

8.13 Pseudo-Random Function (PRF)

PRF is a generic term to describe the chosen hash function (section 8.7 on p.175).

8.13.1 PRF+

$$PRF+(a, b) = T1 \mid T2 \mid T3 \mid \ldots$$

Where

T1 = PRF(a, b | 0x01)

T2 = PRF(a, T1 | b | 0x02)

T3 = PRF(a, T2 | b | 0x03)

Equation 18

PRF+ differs from PRF that PRF+ allows concatenation (represented by the notation |) (section 8.2 on p.164).

8.13.2 Replay Attacks

Replay attacks [99] involve the re-transmission of a legitimate message by an attacker, in order to fool a peer to repeat processing the legitimate message, or to confuse the peer with duplicated legitimate messages. Note that if a message is significantly delayed, the peer may treat the delayed message as a replay attack attempt. Replay attacks can be prevented by including sequence numbers (section 8.18 on p.194) or nonces (section 8.10 on p.178) in the messages.

8.14 Rivert Shamir Adelman (RSA) Algorithms

Public key cryptography is an asymmetric approach, in which a pair of keys is used i.e. a public key and a private key. A private key is kept by its owner. The public key is publicly distributed. RSA is a widely used public key algorithm (e.g. supported by all Web browsers). It provides authentication as well as confidentiality protection. DSA is another popular public key algorithm; however, DSA provides authentication only. Encryption can be done with either the private or the public key in RSA.

8.14.1 RSA Private Key Generation

To generate a RSA private key d, the key owner must select:

- p and q (two large secret prime numbers)
- e (a random number)

e is a value that is made public. Although it is random, the choice of e must meet certain mathematical requirements i.e. it must be relevant to Ø:

$$\phi = (p-1)(q-1)$$

Equation 19

The private key d is then calculated:

$$e * d = 1 \bmod \phi$$

Equation 20

This equation represents a modulus, meaning when dividing (e*d) by Ø, the remainder is 1. Therefore, it can be rewrite as:

$$e * d - 1 = k\phi$$

Where k is any integer

Equation 21

Example 3: Given that e = 13, p =43, q = 59, determine the private key d.

Use the Extended Euclidean Algorithm [100] to determine the private key. First, determine the value of ø:

Ø = (p-1)(q-1)

= (43-1)(59-1)

= 2436

	a	q	x	y
1	(1a) 2436	--	1	0
2	(2a) 13	(2q) 187	0	1
3	(3a) 5	(3q) 2	(3x) 1	(3y) -187
4	(4a) 3	1	(4x) -2	(4y) 375
5	2	1	3	-562
6	1	2	-5 (k)	937 (d)

Table 2 – Extended Euclidean Algorithm

The figures in 1x, 1y, 2x, and 2y are fixed. Fill the boxes as follow:

(1a) and (2a): Fill these two boxes with the value of ø and e.

(2q) and (3a): 2436/13 = 187, remainder = 5.

(3q) and (4a): 13/5 = 2, remainder = 3. And so on, until the number in column a does not give a remainder i.e. 2/1 = 2.

(3x): (3x) = (1x) – (2q)(2x)

= 1 – (187)(0)

= 1

(4x): (4x) = (2x) – (3q)(3x)

= 0 – (2)(1)

= -2

(3y): (3y) = (1y) – (2q)(2y)

= 0 – (187)(1)

= -187

(4y): (4y) = (2y) – (3q)(3y)

= 1 – (2)(-187)

= 375

The figures in 6x and 6y are important. According to Equation 21:

$$e * d - 1 = k\phi$$

Re-writing :

$$1 = -k\phi + d * e$$

Equation 22

Fetching the numbers (6x and 6y) from
Table 2:

$$1 = (-5)(2436) + (937)(13)$$

Therefore d = 937.

Note: d must be positive. If d turns out to be negative (i.e. d'), convert it to a positive number:

$$d = d' + \phi$$

Equation 23

8.14.2 RSA Public Key Generation

A RSA public key is the N and e value. Note that during the generation of the private key, the key owner has already selected a value for e. Typical choices of e are 3 or 65,537. The owner must also compute N:

$$N = p * q$$

Equation 24

N and e are distributed as the owner's public key. Note that encryption can be done with *either* of the RSA private or the public key. For instance, a client – upon receiving the public key from an on-line bank – can use the bank's public key to encrypt its end user's private data before sending the data to the bank. The encrypted data can only be decrypted at the bank (since only the bank

owns the private key). The bank may digitally sign some data with its private key i.e. creating a digital signature. Upon receiving the signed data at the client, the client shall use the bank's public key to verify the signed data.

8.14.3 Encryption with a RSA Public Key

To encrypt a plaintext P with a RSA *public* key:

$$C = P^e \bmod N$$

Where

C = ciphertext

P = plaintext

e = public random shared value from key owner

N = public shared value from key owner

Equation 25

Example 4: Given that P = 1819, e = 13, N = 2537, determine the ciphertext C.

$C = P^e \bmod N$

$C = 1819^{13} \bmod 2537$

In other words, *C* is the remainder of $\frac{1819^{13}}{2537}$

C = 2081

Note: In situation in which e is large, it is very difficult to perform calculation with modules. In this example e = 13, and 13 = 1 + 2 + 4 + 6, Equation 25 can be rewritten as follow to ease calculation:

$$C = [(P^1 \bmod N)(P^2 \bmod N)(P^4 \bmod N)(P^6 \bmod N)](\bmod N)$$

Equation 26

8.14.4 Decryption with a RSA Private Key

To decrypt a plaintext P with a RSA *private* key:

$$P = C^d \bmod N$$

Equation 27

8.14.5 The Level of Complexity of RSA Keys and Prime Numbers

A RSA private key (i.e. d) is calculated by using two large prime numbers (p and q), whereas part of a RSA public keys (i.e. N, but not e) is a product of the two large prime numbers (p and q). The level of complexity (and hence security) of RSA keys depends on the difficulty of factoring a *large* integer. In RSA, this large integer is N, which is product of two large prime numbers p and q.

A prime number has a unique feature: it is an integer that can only be divided by 1 and itself. Example prime numbers are 3, 5, 17, 257. According to the Fundamental Theorem of Arithmetic [101], any (positive) integer has a unique prime factorisation: that is, any integer is a product of *only* one fixed set of prime numbers. For example, the integer 65535 is a product of this particular set of prime numbers:

65535 = 3 x 5 x 17 x 257

Currently, no efficient factorisation algorithm is available for factorising large integers: a recent experiment shown it took 18 months to factorise a 200-digit number into two 100-digit prime numbers [102]. A RSA public key (N but not e) is a product of two large prime numbers (N = p x q). Note that p and q are also used for creating a RSA private key, which must be kept secret at all time. Thus, RSA keys are complex and secured as long as *no* efficient factorisation algorithm exists. Thus, as long as N is large enough, it would be difficult to factorise N (i.e. difficult to obtain p and q), hence it is difficult to obtain b (i.e. the

private key, which is calculated by using on p and q in an one-way function).

It should be noted that the RSA private key (d) must be kept secured. The knowledge of d would enable efficient factorisation of N. For more detail, refer to [95].

The choice of the value for e is another important security factor of RSA (and the performance of RSA operation). e must be a large value. A small value of e leads to weakness in RSA keys. According to Equation 25:

$$C = P^e \bmod N$$

Essentially, the e^{th} root of C is the plaintext P. If e = 3, then the plaintext can be deduced from:

$$P = \sqrt[3]{C \bmod N}$$

Equation 28

Thus, the value of e should be large. A detailed discussion of the choice of e and the impacts of small e value on the RSA algorithm can be found at [95]. However, a large value of e means that more computational time is needed for RSA cryptographic operations: for RSA encryption, the plaintext P goes through a modular exponentiation of e trails. If e and P are large, then the performance overhead of this modular exponentiation process would become undesirable.

8.15 Responder

A responder responds to a SA negotiation instantiated by an Initiator.

8.16 Security Associations (SA)

A Security Association (SA) [103] is a contract defining a set of security parameters [60] to be used between two IPSec hosts. Each IPSec host

maintains a set of associated SAs in its own Security Association Database (SADB)[18]. An SA is simplex i.e. one-way. An IPSec host must define a SA for its outgoing IPSec channel and incoming IPSec channel respectively. For instance, in Figure 52, two IPSec tunnels were established. A particular set of cryptographic key(s) will be used between $SA1_{out}$ and $SA1_{in}$ of host A and host B respectively. Whereas another set of cryptographic key(s) will be used between $SA2_{in}$ and $SA2_{out}$ of host A and host B respectively.

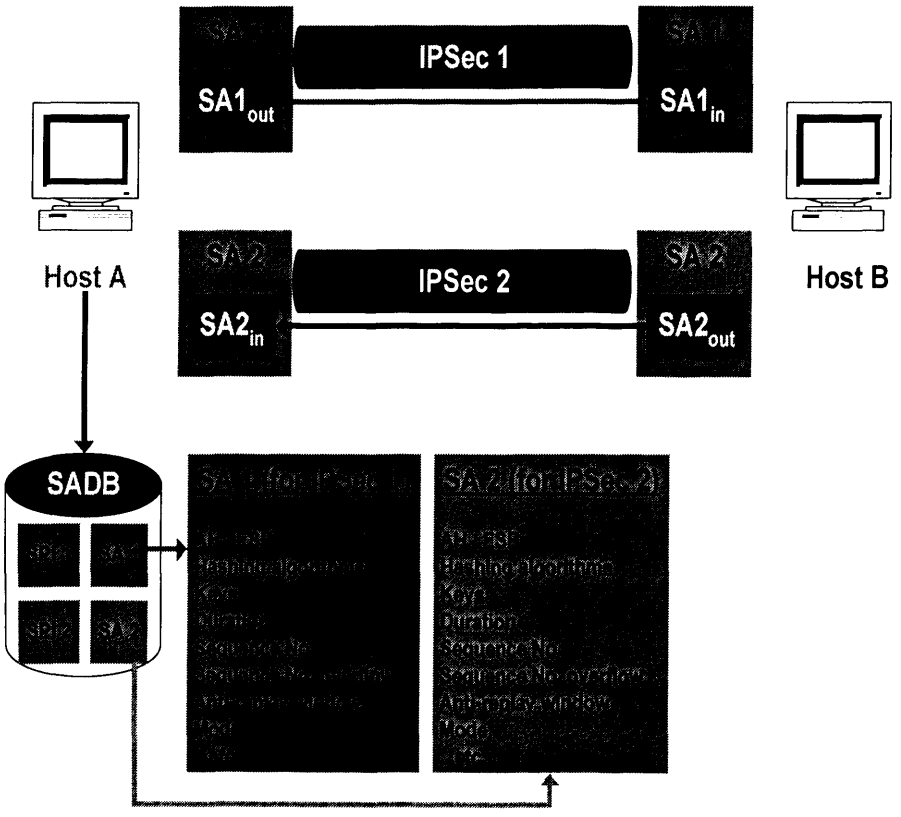

Figure 52 – Simplex SAs

Essential IPSec SA parameters are:

- The IPSec protocol to be used i.e. Authentication Header (AH), Encapsulating Security Payload (ESP), or both.

[18] For simplicity, the SADB of host B is not shown in Figure 52. In this example, only two nodes are involved. The SAs stored in host B's SADB are identical to those stored in host A's SADB.

- The hashing algorithms to be used.
- The keys to be used.
- The duration whilst the key remains valid... and more.

SA parameters are categorised into *protocol-specific* and *generic* fields. The latter field is used by both AH and ESP and are discussed below. Note that some of these fields can be updated in the SAs when necessary.

- Sequence number field

This field contains a 32-bit number in both the AH and ESP header for detecting replay attacks. This number is initially set to zero when a SA is established, and is incremented by one each time the SA is used to secure a packet. The SA should be replaced when this number reaches 4G i.e. 4,000,000,000.

- Sequence number overflow

This field is set (during outbound processing of IPSec packets) to indicate the sequence number has reached 4G.

- Anti-replay window

This field is used during inbound processing to overcome replay attacks.

- Lifetime

This field defines the lifetime of a SA. Lifetime is defined either in 1) number of bytes to be secured by this SA, or 2) the duration of which the SA is valid.

- Mode

Three different modes: tunnel, transport, or wild card. Wild card indicates this SA supports both tunnel and transport mode.

- Tunnel destination

The destination address of tunnelled IPSec packet. Appears as clear text in the header.

- PMTU parameters

Path Maximum Transfer Unit (PMTU) parameters are important when using IPSec tunnel mode. To avoid packet fragmentation, a peer discovers the PMTU of a particular path, and never transmits a packet that exceeds the PMTU.

8.17 SA Creation and Removal

The creation of SA is sub-divided into the negotiation of SA parameters and the insertion of SA into the SADB. Key negotiation between participating peers is a crucial step in the first stage. There are two ways of key negotiation:

- Manual key negotiation

Manual key negotiation is ideal for small-scale IPSec deployment. It is usually done off-line. Manually negotiated keys may run indefinitely – until manually removed.

- IKE (Internet Key Exchange)

IKE is more suitable for large-scale IPSec deployment. IKE meant to work under the guidance of security policies. For instance, a policy may require a particular connection to be secured. The IPSec kernel will then invoke IKE. IKE will then negotiate with the receiver the SA parameters, and then create the SA. The freshly created SA will be added to the sender's SADB.

The removal of SA from SADB can also be performed manually or through IKE. Frequent updating of SA (i.e. remove and create) is essential in order to minimise the chances of keys being compromised. To avoid disruption of an established IPSec communication, a replacement SA (for that particular IPSec communication) is negotiated before the existing SA is deleted.

8.17.1 Security Parameter Index (SPI)

Once a SA has been established between two nodes, packets can be sent

securely between the two nodes by using the established SA. However, a node may have established multiple SAs with different nodes in the network; thus, prior to sending across a packet, there is a need to identify which SA to be used between the sending node and the receiving node [104]. The sender (also known as the Initiator in this thesis) uses the *selectors* to identify the SA. IPSec selectors are:

- Source address and source (IPSec) application port number; destination address and destination (IPSec) application port number.
- Participating specific protocol (of OSI layer 4 or above).
- Name of the policy associated to a user or system.

The receiver (also known as the Responder), however, cannot do so. This is because some of the fields in the packet header (where the selectors are kept) belong to the transport layer. In IPSec, transport layer fields are encrypted. Thus, the receiver would not have access to all the fields (i.e. the selectors) unless the receiver knows the corresponding SA to be used for decryption. The problem becomes a chicken-and-egg problem.

Consequently, at the receiver, each SA is identified by a unique 32-bit Security Parameter Index (SPI). The sender uses the selectors to uniquely index a SA into its SADB. The SA is associated with a unique SPI. The SPI is kept in the AH or ESP header and is sent along with all protected packets from the sender to the receiver. The receiver uses the SPI to retrieve the corresponding SA (from its SADB), and the SA is used in order to process incoming protected packets.

The <SPI, dst> to SA mapping must be unique at all time. Uniqueness of SPI is guaranteed by the host that assigns the SPI. In the case when the sender has

more than one source address, i.e. it has more than one interface, the <SPI, src, dst> combination can be used. The SPI may be re-used when the corresponding SA expires. SPI is included in every IKEv2 message. In IKEv1, *cookies* (section 8.3 on p.164) were used instead of SPI.

8.18 Sequence Number

Sequence numbers of messages are important for anti-*replay attacks* (section 8.13.2 on p.183). If a series of messages are to be exchanged between two peers, each exchanged message must contain a unique sequence number. If same sequence number is found on two messages, the messages are duplicated.

In TCP, for example, sequence numbers are included in all TCP packets. The sequence number of the first TCP message of a particular TCP connection is a pseudo-random number. Each time a message is sent or received during the same connection, the sequence number is incremented by one. A peer would be able to detect replay attacks when two messages arrive with the same sequence number. Note that the sequence number must be pseudo-random, so that the start-off value for each TCP connection is different. This is important because, if the first sequence number of all TCP connections always start off from a fixed value, then if two TCP connections are simultaneously running, there are chances that two TCP packets (of the two TCP connections respectively) may arrive at the same time with the same sequence number. The peer would mis-understand, and thought one of the messages was a replay attack. By using random values, this problem would not happen.

In IKE and IPSec, sequence numbers are known as message IDs. The message ID field in IKE messages is authenticated and integrity protected by

using appropriate keys from the shared key set for added security (section 2.10 on p.55).

9 References

[1] D. Tennenhouse, D. Wetherall, "Towards an Active Network Architecture" in Multimedia Computing and Networking, San Jose, California, USA, Jan, 1996.

[2] J. Vicente, H. Cartmill, G. Maxson, S. Siegel, and R. Fenger, "Managing Enhanced Network Services: a Pragmatic View of Policy-based Management", Intel Technology Journal, 1st Quarter, 2000.

[3] M. Brunner, "A Service Management Toolkit for Active Networks", in IFIP/IEEE International Symposium on Network Operations and Management (IFIP/IEEE-NOMS), Hawaii, USA, April 2000, pp. 265-278.

[4] The DARPA Active Networks web site, http://nms.lcs.mit.edu/darpa-activenet/

[5] L. Peterson, et al., "NodeOS Interface Specification", AN NodeOS Working Group, Jan 2001, http://protocols.netlab.uky.edu/~calvert/nodeos-latest.ps

[6] S. Murphy, et al., "Security Architecture for Active Networks", AN Security Working Group, July 2001, http://protocols.netlab.uky.edu/~calvert/sec-latest.ps

[7] The AN Composable Services Working Group, "Composable Services for Active Networks", Georgia Institute of Technology, Sep 1998.

[8] K. Calvert, et al., "Architectural Framework for Active Networks", Active Network Working Group, July 1999, http://protocols.netlab.uky.edu/~calvert/arch-docs.html

[9] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, G. Minden, "A Survey of Active Network Research", in IEEE Communications Magazine, Vol. 35, Issue 1, Jan 1997, pp. 80-86.

[10] J. Kornblum, D. Raz, Y. Shavitt, "The Active Process Interaction with its Environment", in Computer Networks, the International Journal of Computer and Telecommunications Networking, Vol. 36, Issue 1, June 2001, pp. 21-34, ISSN: 1389-1286

[11] D. Alexander, "ALIEN: A Generalised Computing Model of Active Networks", PhD thesis, University of Pennsylvania, Feb 1997.

[12] R. Smith, "Authentication, From Passwords to Public Keys", Addison-Wesley, 2002, ISBN: 0-201-61599-1, p. 143-148.

[13] The Future Active IP Networks (FAIN) Project web site, http://www.ist-fain.org

[14] ANEP: Active Network Encapsulation Protocol, http://www.cis.upenn.edu/~switchware/ANEP/

[15] D. Alexander, B. Braden, C. Gunter, A. Jackson, A. Keromytis, G. Minden, D. Wetherall, "Active Network Encapsulation Protocol (ANEP)", Active Network Groups, DRAFT RFC (obsolete), temporarily available at: http://www.ee.ucl.ac.uk/~lcheng/Papers/Draft-RFC-ANEP.txt

[16] J. Moore, M. Hicks, S. Nettles, "Practical Programmable Packets", in proceedings of 12th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Anchorage, AK, USA, April 2001, pp. 41-50.

[17] T. Suzuki, C. Kitahara, S. Denazis, L. Cheng, W. Eaves, A. Galis, T. Becker, D. Gabrijelcic, A. Lazanakis, G. Karetsos, "Dynamic Deployment & Configuration of Differentiated Services Using Active Networks", IFIP TC6 5th International Workshop on Active Networks (IWAN), Kyoto, Japan, Dec 2003, pp. 190-201.

[18] L. Peterson, Y. Gottlieb, M. Hibler, P. Tullmann, J. Lepreau, S. Schwab, H. Dandekar, A. Purtell, J. Hartman, "An OS Interface for Active Routers", IEEE Journal on Selected Areas in Communications, Vol. 19, No. 3, March 2001.

[19] J. Smith, K. Calvert, S. Murphy, H. Orman, L. Peterson, "Activating Networks: A Progress Report", IEEE Computer, Vol. 32, pp.32-41, April 1999.

[20] P. Tullmann, M. Hibler, J. Lepreau, "Janos: A Java-Oriented OS for Active Network Nodes", IEEE Journal on Selected Areas in Communications, Vol. 19, No. 3, March 2001.

[21] S. Denazis, S. Karnouskos, T. Suzuki, S. Yoshizawa, "Component-Based Execution Environments of Network Elements and a Protocol for their Configuration", in IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Vol. 34, No. 1, Feb 2004.

[22] M. Hicks, A. Keromytis, J. Smith, "A Secure PLAN", in proceedings of the First International Working Conference on Active Networks (IWAN), volume 1653 of *Lecture Notes in Computer Science*, pages 307-314. Springer-Verlag, June 1999. Reprinted with extensions in DARPA Active Networks Conference and Exposition (DANCE) and IEEE Transactions on Systems, Man, and Cybernetics, Part C.

[23] D. J. Wetherall, J. Guttag, D. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", IEEE Open Architectures and Network Programming (OpenArch), San Francisco, CA, USA, April 1998, pp. 117-129.

[24] L. Cheng, A. Galis, "Security Protocol for Active Networks", in Proceedings of the 14th IEEE International Conference on Networks

(ICON), Singapore, Sep 2006.

[25] L. Cheng, A. Galis, "Simple Key Exchange for Active Networks", in Proceedings of the 13th IEEE International Conference on Networks (ICON), Kuala Lumpur, Malaysia, Nov 2005, Vol. 1, pp. 6-11.

[26] L. Cheng, A. Galis, C. Todd, "Active Network Authentication", in Proceedings of London Communication Symposium (LCS), London, UK, Sep 2003.

[27] S. Murphy, A. Hayatnagarkar, S. Krishnaswamy, W. Morrison, R. Watson, "Prophylactic, Treatment and Containment Techniques for Ensuring Active Network Security", in proceedings of DARPA Information Survivability Conference and Exposition, April 2003, Vol. 1, pp. 97-108.

[28] S. Murphy, E. Lewis, R. Puga, R. Watson, R. Yee, "Strong Security for Active Networks", in proceedings of IEEE Open Architectures and Network Programming (OpenArch), Anchorage, AK, USA, April 2001, pp. 63-70.

[29] R. Smith, "Authentication, From Passwords to Public Keys", Addison-Wesley, 2002, ISBN: 0-201-61599-1, p. 234-239.

[30] C. Kaufman, "RFC 4306- Internet Key Exchange (IKE v2) Protocol", Network Working Group, Request for Comments 4306, Dec 2005.

[31] D. Harkins, D. Carrel, "RFC – 2409 The Internet Key Exchange (IKE)", Network Working Group, Request for Comments 2409, Nov 1998.

[32] D. Piper, et. Al., "The Internet IP Security Domain of Interpretation for ISAKMP", IETF RFC 2407, Nov 1998.

[33] D. Maughan, M. Schertler, M. Schneider, J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", IETF RFC 2408, Nov 1998.

[34] H. Soussi, M. Hussain, H. Afifi, D. Seret, "IKEv1 and IKEv2: A Quantitative Analyses", in Proceedings of World Enformatika Society (WEC), Istanbul, Turkey, Jun 2005, Vol. 6, pp. 194-198.

[35] K. Matsuura, H. Imai, "Resolution of ISAKMP/Oakley Key-Agreement Protocol Resistant Against Denial-of-Service Attack", in Proceedings of Internet Workshop (IW), Piscataway, New Jersey, USA, Feb 1999, pp. 17-24.

[36] R. Perlman, C. Kaufman, "Key Exchange in IPSec: Analysis of IKE", in IEEE Internet Computing, Vol. 4., Issue 6, Dec 2000, pp. 50-56

[37] R. Smith, "Authentication, From Passwords to Public Keys", Addison-Wesley, 2002, ISBN: 0-201-61599-1, p. 401-432.

[38] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", Federal Information Processing Standards Publication (FIPS PUB) 186, May 1994.

[39] R. Smith, "Authentication, From Passwords to Public Keys", Addison-Wesley, 2002, ISBN: 0-201-61599-1, p. 373-382.

[40] D. Salomon, "Data Privacy and Security", Springer-Verlag, New York, 2003, ISBN: 0-387-00311-8, pp.195-202.

[41] R. Smith, "Authentication, From Passwords to Public Keys", Addison-Wesley, 2002, ISBN: 0-201-61599-1, p. 369-372.

[42] N. Doraswamy, D. Harkins "IPSec The New Security Standard for the Internet, Intranets, and Virtual Private Networks", 2nd edition, Prentice Hall, 2003, ISBN: 0-13-046189-X, pp. 239-244.

[43] D. S. Alexander, W. Arbaugh, D. Keromytis, J. Smith, "A Secure Active Network Environment Architecture: Realization in SwitchWare", in IEEE Network Magazine (special issue on Active and Controllable Networks), June 1998.

[44] W. Arbaugh, A. Keromytis, D. Farber, J. Smith, "Automated Recovery in a Secure Bootstrap Process", in proceedings of Network and Distributed System Security Symposium, Internet Society, March 1998, pp.155-167.

[45] H. Krawczyk, M. Bellare, R. Canetti, "RFC 2104 – HMAC: Keyed-Hashing for Message Authentication", Network Working Group, Request For Comments: 2104, February 1997.

[46] T. Becker, L. Cheng, S. Denazis, D. Gabrijelcic, A. Galis, "Management and Performance of Virtual and Execution Environments in FAIN", in Proceedings of the 6th International Working Conference on Active Networks (IWAN), Kansas, USA, Oct 2004.

[47] T. Becker, L. Cheng, S. Denazis, D. Gabrijelcic, A. Galis, G. Karetsos, A. Lazanakis, "FAIN: A Flexible Node Architecture for the Dynamic Deployment of New Services", in Proceedings of the 6th International Working Conference on Active Networks (IWAN), Kansas, USA, Oct 2004.

[48] W. Eaves, L. Cheng, A. Galis, T. Becker, T. Suzuki, S. Denazis, C. Kitahara, "SNAP based Resource Control for Active Networks", in Proceedings of IEEE Global Telecommunications Conference (Globecom), Taipei, Taiwan, Nov 2002.

[49] S. Krishnaswamy, J. Evans, G. Minden, "A Prototype Framework for providing Hop-by-Hop Security in an Experimentally Deployed Active Network", in proceedings of the IEEE DARPA Active Networks Conference and Exposition (DANCE), San Francisco, CA, USA, May 2002, pp. 216-222.

[50] P. Pessi, "Secure Multicast", in Proceedings of Helsinki University of Technology Seminar on Network Security, 1995, http://www.tml.tkk.fi/Opinnot/Tik-110.501/1995/multicast.html#intro

[51] MSEC (Multicast Security) Working Group, http://www.securemulticast.org/msec-index.htm

[52] N. Doraswamy, D. Harkins "IPSec The New Security Standard for the Internet, Intranets, and Virtual Private Networks", 2nd edition, Prentice Hall, 2003, ISBN: 0-13-046189-X, pp. 220-233.

[53] T. Aurish, C, Karg, "Using the IPSec Architecture for Secure Multicast Communication", extended abstract for the 8th International Command and Control Research and Technology Symposium (ICCRTS), Washington D.C., USA, Jun 2003, pp. 14-17, http://www.dodccrp.org/events/2003/8th_ICCRTS/pdf/027.pdf

[54] R. Smith, "Authentication, From Passwords to Public Keys", Addison-Wesley, 2002, ISBN: 0-201-61599-1, p. 240-241.

[55] C. Kaufman, "RFC 4306- Internet Key Exchange (IKE v2) Protocol", Network Working Group, Request for Comments 4306, Dec 2005, pp. 20.

[56] C. Kaufman, "RFC 4306- Internet Key Exchange (IKE v2) Protocol", Network Working Group, Request for Comments 4306, Dec 2005, pp. 19-20, http://www.rfc-archive.org/getrfc.php?rfc=4306

[57] W. Aiello, S. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. Keromytis, O. Reingold, "Efficient, DoS-Resistant Secure Key Exchange for Internet Protocols", in proceedings of the 9th ACM Computers and Communications Security Conference (CCS), Washington D.C., USA, 2002, pp. 48-58.

[58] N. Doraswamy, D. Harkins "IPSec The New Security Standard for the Internet, Intranets, and Virtual Private Networks", 2nd edition, Prentice Hall, 2003, ISBN: 0-13-046189-X, pp. 56-57.

[59] N. Doraswamy, D. Harkins "IPSec The New Security Standard for the Internet, Intranets, and Virtual Private Networks", 2nd edition, Prentice Hall, 2003, ISBN: 0-13-046189-X, pp. 122-123.

[60] N. Doraswamy, D. Harkins "IPSec The New Security Standard for the Internet, Intranets, and Virtual Private Networks", 2nd edition, Prentice Hall, 2003, ISBN: 0-13-046189-X, pp. 71-77.

[61] N. Doraswamy, D. Harkins "IPSec The New Security Standard for the Internet, Intranets, and Virtual Private Networks", 2nd edition, Prentice Hall, 2003, ISBN: 0-13-046189-X, pp. 59-71.

[62] S. Kent, R. Atkinson, "RFC 2401 – Security Architecture for the Internet Protocol", Network Working Group, Request for Comments 2401, Nov

1998.

[63] N. Doraswamy, D. Harkins "IPSec The New Security Standard for the Internet, Intranets, and Virtual Private Networks", 2nd edition, Prentice Hall, 2003, ISBN: 0-13-046189-X, pp. 43-54.

[64] C. Kaufman, "RFC 4306- Internet Key Exchange (IKE v2) Protocol", Network Working Group, Request for Comments 4306, Dec 2005, pp. 34, http://www.rfc-archive.org/getrfc.php?rfc=4306

[65] Juniper Networks, "The E-Series Routing Protocols Configuration Guide, Vol. 1", http://www.juniper.net/techpubs/software/erx/erx51x/swconfig-routing-vol 1/html/ipsec-config5.html

[66] N. Doraswamy, D. Harkins "IPSec The New Security Standard for the Internet, Intranets, and Virtual Private Networks", 2nd edition, Prentice Hall, 2003, ISBN: 0-13-046189-X, pp. 123-127

[67] H. Krawczyk, et al., "SIGMA: the SIGn-and-Mac Approach to Authenticated Diffie-Hellman and its Use in the IKE Protocols", in proceedings of Advances in Cryptography (CRYPTO), LNCS 2729, Springer, 2003, http://www.ee.technion.ac.il/~hugo/sigma.html

[68] C. Kaufman, "RFC 4306- Internet Key Exchange (IKE v2) Protocol", Network Working Group, Request for Comments 4306, Dec 2005, pp. 88, http://www.rfc-archive.org/getrfc.php?rfc=4306

[69] C. Kaufman, "RFC 4306- Internet Key Exchange (IKE v2) Protocol", Network Working Group, Request for Comments 4306, Dec 2005, pp. 27-29, http://www.rfc-archive.org/getrfc.php?rfc=4306

[70] Internet mailing list, "UDP DoS attack in Win2K via IKE", http://marc.theaimsgroup.com/?l=bugtraq&m=100774842520403&w=2

[71] Internet references, "Nortel VPN Router Malformed IKE Packet DoS", http://www.osvdb.org/16918

[72] P. Eronen, "Denial of service in public key protocols", in proceedings of the Helsinki University of Technology Seminar on Network Security, December 2000, http://www.niksula.hut.fi/~peronen/publications/netsec_2000.pdf

[73] P. Karn, W. Simpson, "Photuris: Session-Key Management Protocol", IETF RFC 2522, March 1999.

[74] K. Matsuura, H. Imai, "Modification of Internet Key Exchange Resistant against Denial-of-Service", in proceedings of Internet Workshop (IWS), Feb 2000, pp.167-174.

[75] Y. Zheng, "Digital Signcryption or How to Achieve Cost(Signature &

Encryption) << Cost(Signature) + Cost(Encryption)", in Proceedings of Advances in Cryptology (CRYPTO), Berlin, Germany, Springer-Verlag, LNCS 1294, August 1997, pp.165-179.

[76] D. Coppersmith, M. Franklin, J. Patarin, M. Reiter, "Low-exponent RSA with related messages", in Proceedings of Advances in Cryptology (EUROCRYPT), vol. 1070 of LNCS, Springer-Verlag, Saragossa, Spain, May 1996, pp. 1-9.

[77] A. Odlyzk, "The future of integer factorisation", CryptoBytes 1(2):5—12, 1995.

[78] D. Cooper, "A Model of Certificate Revocation", in Proceedings of the 15th IEEE-Annual Computer Security Applications Conference (ACSAC), Scottsdale, AZ, USA, Dec 1999, http://csrc.nist.gov/pki/documents/acsac99.pdf

[79] W. Diffie, P. Oorschot, M. Wiener, "Authentication and Authenticated Key Exchanges", Designs Codes and Cryptography, vol. 2, 1992, pp. 107-125.

[80] B. Springer, L. Kilmartin, "Performance Evaluation of the IKE Protocol under Dynamic VoIP Network Conditions", in proceedings of the Irish Signals and Systems Conference, Limerick, Ireland, 2003.

[81] SonicWall, "Creating IKE IPSec VPN Tunnels between SonicWALL Devices and Cisco 3000 VPN Concentrators", TECHnotes, May 2002, http://www.vpn-technology.com/Interoperability/SonicWALL%20VPN%20with%20Cisco%203000.pdf

[82] Verisign, http://www.verisign.com

[83] Algorithm Solutions Software GmbH, "Key for Cryptography (Crypt Key)", The LEDA menu, http://www.algorithmic-solutions.info/leda_manual/CryptKey.html#CryptKey

[84] D. Salomon, "Data Privacy and Security", Springer-Verlag, New York, 2003, ISBN: 0-387-00311-8, pp.170-172.

[85] L. Cheng, K. Jean, R. Ocampo, A. Galis, "Towards Flexible Service-aware Adaptation Management in Ambient Networks", to appear in Proceedings of the 1st IEEE International Conference on Networks (ICON), Singapore, Sep 2006.

[86] L. Cheng, R. Ocampo, K. Jean, A. Galis, C. Simon, R. Szabo, P. Kersch, R. Giaffreda, "Towards Distributed Management Systems Composition & Decomposition in Ambient Networks", to appear in Proceedings of IFIP/IEEE Distributed Systems: Operations and Management (DSOM), Dublin, Ireland, Oct 2006 (acceptance ratio: ~20%).

[87] R. Ocampo, L. Cheng, K. Jean, A. Prieto, A. Galis, "Towards a Context

Monitoring System for Ambient Networks", to appear as work-in-progress report in IEEE Chinacom, Peking, China, Oct 2006.

[88] L. Cheng, R. Ocampo, A. Galis, R. Szabo, C. Simon, P. Kersch, "Self-management in Ambient Networks for Service Composition", in Proceedings of IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM), Montreal, Canada, Oct 2005.

[89] K. Lim, C. Adam, R. Stadler, "Decentralizing Network Management", KTH Technical Report, Nov 2005, available at: http://www.ee.kth.se/~stadler/nmrg/DECENTRALIZING-KTHTR-2005.pdf

[90] K. Lim, R. Stadler, "Real-time Views of Network Traffic using Decentralised Management", 9th IFIP/IEEE International Symposium on Integrated Network Management (IM) 2005, Nice, France, May 2005.

[91] Public Key Infrastructure (X.509) Working Group, http://www.ietf.org/html.charters/pkix-charter.html

[92] Sun Microsystems, "keytool – key and certificate Management Tool", http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html

[93] Microsoft, "Cryptographic Key Storage and Exchange", MSDN Library, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/seccrypto/security/cryptographic_key_storage_and_exchange.asp

[94] P. Karn, W, Simpson, "The Photuris Session Key Management Protocol", Internet Draft, draft-ietf-ipsec-photuris-03.txt, Sep 1995 (obsolete).

[95] RSA Laboratories, "Chapter 3.1, Techniques in Cryptography", Crypto FAQ, http://www.rsasecurity.com/rsalabs/node.asp?id=2215

[96] D. Salomon, "Data Privacy and Security", Springer-Verlag, New York, 2003, ISBN: 0-387-00311-8, pp.201.

[97] P. Oorshot, M. Wiener, "On Diffie-Hellman Key Agreement with Short Exponents", in proceedings of Eurocrypt 1996, LNCS 1070, Springer-Verlag, 1996, http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/E96/332.PDF

[98] RSA Laboratories, "Chapter 3.6, Techniques in Cryptography", Crypto FAQ, http://www.rsasecurity.com/rsalabs/node.asp?id=2258

[99] R. Smith, "Authentication, From Passwords to Public Keys", Addison-Wesley, 2002, ISBN: 0-201-61599-1, p. 338-339.

[100] B. Ikenaga, "An Example Using the Extended Euclidean Algorithm", Internet article, http://www.millersv.edu/¬bikenaga/absalg/exteuc/exteucex.html

[101] H. Thomas, "A History of Greek Mathematics", Vol. 2, Dover Publications,

new edition, ISBN: 0-486-24073-8.

[102] F. Bahr, M. Boehm, J. Franke, T. Kleinjung, e-mail announcement on RSA factorisation, http://www.crypto-world.com/announcements/rsa200.txt

[103] N. Doraswamy, D. Harkins "IPSec The New Security Standard for the Internet, Intranets, and Virtual Private Networks", 2nd edition, Prentice Hall, 2003, ISBN: 0-13-046189-X, pp. 46-47.

[104] N. Doraswamy, D. Harkins "IPSec The New Security Standard for the Internet, Intranets, and Virtual Private Networks", 2nd edition, Prentice Hall, 2003, ISBN: 0-13-046189-X, pp. 153-159.

[105] M. Wutka, et al., "Hacking Java: The Java Professional's Resource Kit", Chapter 27, Que Pub; ISBN: 078970935X, Nov 1996.

[106] Ius mentis, "PGP Attack FAQ: The asymmetric cipher", Internet references, http://www.iusmentis.com/technology/encryption/pgp/pgpattackfaq/asymmetric/