TAMPERE UNIVERSITY OF TECHNOLOGY
Degree Programme in Information Technology

JUHA KORHONEN

HOST IDENTITY PROTOCOL (HIP) IMPLEMENTATION IN THE SYMBIAN ENVIRONMENT

Master of Science Thesis

Examiner: Prof. Jarmo Harju
Prof. Hannu-Matti Järvinen
Advisors: Dr. Lars Eggert
Dr. Pasi Sarolahti
Examiner and topic approved in
Information Technology
Department Council
Meeting on 10th December 2007

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
KORHONEN, JUHA : Host Identity Protocol (HIP) Implementation in the Symbian Environment
Master of Science Thesis, 70 pages, 7 confidental Appendix pages
May 2008
Major: Embedded Systems
Examiner: Professor Jarmo Harju, Professor Hannu-Matti Järvinen
Keywords: host multihoming, host mobility, Internet topology, Host Identity Protocol, Host Identity namespace, Symbian, IPsec

Host multihoming and mobility are problems in the current Internet. To support host mobility and multihoming, dynamic readdressing is needed. Currently a host cannot change its IP addresses without disconnecting the sockets, which are bound to respective IP addresses. Because of that, applications have to be aware of such mobility and multihoming events. To avoid this overhead complexity in the applications this could be done in the network layer.

Additionally, the current IP namespace does not provide identification of hosts, because it is divided into the public and private IP addresses that limit the uniqueness of IP addresses. Furthermore, an IP address is non-cryptographic and thus does not provide any security between hosts. IP addresses do not guarantee data origin confidentiality.

The Host Identity Protocol (HIP) provides a solution for the introduced problems. HIP is a key negotiation protocol, but additionally, HIP introduces a new namespace, the host identity namespace. The namespace separates IP addresses and the host identifiers. A host identifier is the public key of an asymmetric cryptographic key pair. Thus, a host can be identified and authenticated based on its host identifier.

The scope of the thesis was the implementation of HIP. A prototype handling the Bound End-to-End Tunnel (BEET) mode for the IPsec and the part of the HIP Base Exchange (BEX) messaging is implemented. Most of the complex HIP parameters were not implemented. Secondly, this thesis provides an extensive background for future implementations. The topic is important, because HIP is useful for mobile devices, which have multiple interfaces such as Symbian handsets.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
JUHA KORHONEN: HIP-protokollan implementointi Symbian-ympäristössä
Diplomityö, 70 sivua, 7 luottamuksellista liitesivua
Toukokuu 2008
Pääaine: Sulautetut järjestelmät
Tarkastajat: Professori Jarmo Harju, Professori Hannu-Matti Järvinen
Avainsanat: käyttäjän liikkuvuus, Internet, HIP-protokolla, Host Identity -nimiavaruus, Symbian, IPsec

Käyttäjän liikkuvuus ja rinnakkaisten verkkoyhteyksien yhtäaikainen käyttäminen ovat ongelmia Internetissä. Näiden mahdollistaminen vaatii dynaamista IP-osoitteiden hallintaa. Nykyään käyttäjä ei kykene vaihtamaan IP-osoitetta ilman sokettien uudelleenyhdistämistä, koska soketit ovat sidottuja IP-osoitteisiin. Tämän takia verkkoyhteyksiä ja liikkuvuutta hallitaan usein sovellustasolla, mikä monimutkaistaa turhaan sovelluksia, vaikka kyseiset ongelmat voitaisiin ratkaista verkkokerroksella.

Toisaalta nykyinen IP-nimiavaruus (ts. osoiteavaruus) ei mahdollista käyttäjän tunnistamista, koska nimiavaruus on jaettu julkiseen ja yksityiseen osioon, minkä seurauksena IP-osoiteet eivät ole välttämättä yksiselitteisiä. Sen lisäksi IP-osoite ei ole kryptografisesti muodostettu eikä mahdollista minkäänlaista tietoturvaa käyttäjien välillä. IP-osoitteet eivat takaa luottamuksellisesti datan alkuperää.

HIP-protokolla ratkaisee edellä mainitut ongelmat. HIP on avaintenvaihtoprotokolla, mutta sen lisäksi se määrittelee uuden nimiavaruuden, joka perustuu käyttäjän identiteetteihin (Host Identity). Nimiavaruus erottaa käyttäjän identiteetin IP-osoitteista. Host Identifier on epäsymmetrisen avainparin julkinen avain, ja tästä johtuen sitä voidaan käyttää käyttäjän tunnistamiseen ja autentikointiin.

Diplomityö käsittelee HIP-protokollaa ja sen implementointia. Työssä on toteutettu prototyyppi, joka pitää sisällään IPsecin BEET moodin, ja avaintenvaihdon perusteet. Monimutkaisempia HIP-parametreja työssä ei ole toteutettu. Tämän lisäksi tarkoituksena oli tarjoa informaatiota tulevaisuuden projekteja varten. Diplomityö tarjoaa pohjan jatkokehitykselle. Aihetta voidaan pitää erittäin tärkenä, koska HIP on hyödyllinen mobiilipäätelaitteille, jotka voivat kiinnittyä useaan eri verkoon ja joilla on siksi useampia erilaisia verkkorajapintoja käytössään samanaikaisesti.

PREFACE

This Master of Science thesis was written as a part of the Nokia Research Center mobility and multihoming research done in the Future Internet team. The work was started during summer 2007 and the project was finished in spring 2008. The thesis has also a practical part, an implementation.

First of all, I want to thank Dr. Pasi Sarolahti and Dr. Lars Eggert. They have been giving me crucial guidance to get this thesis written. I would like to thank Professor Jarmo Harju for his guidance, useful comments, and writing tips. I want to thank Professor Hannu-Matti Järvinen for giving his contribution for this thesis.

Last but not least, I would like to thank my girlfriend Jutta, my family and everyone at Nokia for making this possible.

Otaniemi, May 5, 2008

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS AND NOTATIONS

ACL	Access Control List
AH	Authentication Header
API	Application Programming Interface
ARPANET	Advanced Research Projects Agency Network
BEET	Bound End To End Tunnel
BEX	Base Exchange
CBC	Cipher Block Chaining
CIDR	Classless Inter Domain Routing
CPM	Comms Provider Module
CPU	Central Processing Unit
DES	Data Encryption Standard
DH	Diffie-Hellman
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DNSSEC	DNS Security Extensions
DoS	Denial of Service
DSA	Digital Signature Algorithm
ESP	Encapsulation Security Payload
FQDN	Fully Qualified Domain Name
IETF	Internet Engineering Task Force
HI	Host Identifier
HIP	Host Identity Protocol
HIT	Host Identity Tag
HMAC	Hash Message Authentication Code
HTTP	Hypertext Transfer Protocol
ICV	Integrity Check Value
ICMP	Internet Control Message Protocol
ICMPv4	Internet Control Message Protocol version 4
ICMPv6	Internet Control Message Protocol version 6
IAP	Internet Access Points
IKE	Internet Key Exchange protocol
IP	Internet Protocol
IPC	Inter Process Communication
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6

IPsec	Internet Protocol security
LAN	Local Area Network
LSI	Local Scope Identifier
MAC	Message Authentication Code
MD5	Message Digest algorithm 5
MITM	Man In The Middle
NAI	Network Access Identifier
NAT	Network Address Translation
OO	Object Oriented
OS	Operating System
OSI	Open Systems Interconnection
PDA	Personal Digital Assistant
PDU	Protocol Data Unit
PKC	Public Key Cryptography
POSIX	Portable Operating System Interface
PPP	Point to Point Protocol
QOS	Quality of Service
ROM	Read Only Memory
RSA	Rivest Shamir Adleman algorithm
SA	Security Association
SAD	Security Association Database
SAP	Service Access Point
SHA	Secure Hash Algorithm
SID	Secure Identifier
SIP	Session Initiation Protocol
SP	Security Policy
SPI	Security Parameter Index
SPD	Security Policy Database
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User Interface
UID	Unique Identifier
VID	Vendor Identifier
VPN	Virtual Private Network

1. INTRODUCTION

In the current Internet architecture, a host cannot change its IP address, without breaking the bindings of the upper layers. Sockets that bind the upper layers to the IP address are disconnected if the respective IP address becomes useless or changes.

The upper layer bindings prevent dynamic readdressing in rapid manner because sockets must be first disconnected and then reconnected with the new IP address. The problem becomes significant for hosts, which have multiple IP addresses, or hosts moving in the Internet topology. In the future, the problem becomes especially significant because the amount of mobile hosts such as laptops, Personal Digital Assistants (PDAs) and mobile phones is increasing. For example, when a person moves with the mobile host geographically from work to home, more precisely from network to another, the readdressing has to be done every time in order to stay connected.

Host mobility and end-host multihoming supports seamless change of an IP address used by a host. Currently the functionality that handles host mobility and end-host multihoming must be done partly in the application layer. That is not a good solution, because of the disconnection of a socket and delay caused by the reconnection of the socket. It is not reasonable to have such complexity in every application. In addition, the connectivity of a host, i.e. ability to use network to send and receive information, may change over time. The connection may lose some bandwidth or the link attached may be down or congested. In these cases, the host may want to change the used link.

On the other hand, IP addresses and domain names are currently all that we have, and we do too much with them. Semantic overloading and extended functionality are increasingly complicating the IP namespace [Mos06, page 6] [Ste96, page 429]. IP address has two roles: an identifier of a host and a topological information string. Extended functionality such as Network Address Translators (NATs), divides the address namespace to the public and private namespaces that restricts identification and the validity of the topological information to the private address space.

In addition, the current namespaces, IP and Domain name, do not provide unique authentication of hosts or datagrams [Mos06, page 6]. Besides that, the namespace does not provide anonymity of hosts [Mos06, page 6]. A host cannot be authen-

ticated based on the IP address, because an IP address may change over time. Anonymity requires that a host cannot be identified. Because the Domain Names are associated with several other systems such as Email, Hypertext Transfer Protocol (HTTP), and Session Initiation Protocol (SIP) there is no anonymity in Domain names. These matters have several consequences. Because authentication is lacking, reliable communication must occur by other means. Anonymity, on the other hand, improves both privacy and security. Privacy can be achieved if a host can keep its identity secret from the unwanted parties. If privacy can be achieved security is also improved. A malicious eavesdropper or an unwanted peer cannot determine the identity without the help of the respective host.

The Host Identity Protocol (HIP) provides a solution for the introduced problems. First, it supports host mobility and host multihoming. It defines a new namespace, Host Identity namespace, and sockets can bind to that [Mos06, pages 8-9, 11]. Instead, the bindings of upper layers do not break if IP addresses change. When the IP address of a network interface changes, a socket stays bound to a Host Identifier (HI), which is a concrete representation of a Host Identity. Second, the new HI namespace provides identification of hosts that decreases the semantic overloading of the IP addresses in the upper layers. A host can be identified based on the HI. Third, Host Identifiers are the public keys of an asymmetric key pair. The authentication of a host is based on the Host Identifier. Finally, a Host Identity can provide anonymity for a host, if the Host Identity is not published.

1.1 Scope

Symbian does not have native implementation for HIP. The fact that many mobile devices use the Symbian platform makes it important to solve host mobility and host multihoming issues on the platform, because mobile devices roam between networks. Additionally, HIP may be globally deployed in the future and then it should be supported by every host.

This thesis work implements parts of HIP and evaluates protocol suitability for the Symbian environment. The implementation is a prototype. The thesis concentrates on HIP on the Symbian platform version 9.2. An implementation is designed and the implemented parts are tested in the Symbian platform. The implementation is built as a modular software entity to be able to extend it with HIP properties introduced in other drafts by Internet Engineering Task Force (IETF). The design, implementation, and testing covers relevant parts of IPsec BEET draft [Nik07b], HIP architecture [Mos06], HIP Base Exchange [Mos07]. In other words, the implementation includes the part of the HIP Base Exchange, and implements a new form

of IPsec Encapsulation Security Payload (ESP) encapsulation called Bound-End-to-End-Tunnel (BEET). The Symbian implementation provides valuable information about the HIP protocol and about Symbian as a networking environment. The work also provides valuable information to Nokia about bringing HIP to mobile devices in the future.

1.2 Organization of the thesis

The thesis is organized as follows. Chapter 2 describes the background of the Host Identity Protocol. It first introduces several related concepts, such as IP addresses and Internet Protocol Security and then moves on to describing the HIP protocol. Chapter 3 reviews the Symbian Operating System, especially its networking features. Essential software concepts such as Active Objects, Client-Server, Symbian Socket Application Programming Interface (API) etc. are described. Chapter 4 concentrates on the design aspects of the implemented prototype. Host Identity Protocol and software design are described. The structure of the system is represented in Chapter 4. Chapter 5 describes the details of the implementation. The focus is on the functional description of the system. Chapter 6 discusses and analyzes the implementation. Finally, Chapter 7 summarizes the thesis. Some parts of the thesis are confidential and presented in the confidential appendices.

2. BACKGROUND

This chapter introduces the current Internet architecture at the network layer and describes the role of IP addresses as locators and identifiers. Second, it focuses on end-host multihoming and host mobility concepts. Next, this chapter concentrates on the security architecture for the Internet Protocol (IPsec), especially how it supports the design of the Host Identity Protocol. Then, it illustrates the Host Identity Protocol at a high level. HIP has a solution for the introduced problems with the current IP address space, hence referred as *IP namespace*. HIP introduces a new namespace, which solves the problems with the many roles of the IP addresses and semantic overloading of them. Finally, this chapter introduces the new cryptographic namespace, which adds a new layer to the OSI reference model [Zim80]. Chapter 4 illustrates HIP in more detail.

2.1 Internetworking

When the architecture of the Internet and its predecessor was considered, the design was made with prevailing needs in mind. The design did not take future demands, address space exhaustion combined with exponentially growing number of hosts and networks, into account. The need to solve arising problems such as congestion of the network led to new protocol design. According to the TCP/IP model, TCP [Pos81b] was introduced as a transport layer protocol and Internet Protocol (IP) [Pos81a] as a network layer protocol. This original IP protocol is still a crucial part of the internetworking scheme; although a new IP version 6 [Dee95], has been introduced.

The Internet architecture has evolved from the ARPANET [Rob67], which was an experimental packet-switched network in the United States. In *packet-switched* networks, the *path* i.e. the way a packet travels, is not reserved during the communication [Kur07, page 22]. In a packet-switched network, each packet is routed individually. Earlier in telecommunications, circuit-switched networks were in use. *Circuit-switching* establishes a fixed bandwidth circuit on the path before using it. The ARPANET connected individual networks, which were implemented with different technologies, and established a packet-switched internetwork. The Internet Protocol (IP) [Pos81a] was introduced as a common protocol for the ARPANET.

The *Internet Protocol* provides a best effort service for packet delivery, i.e. it did not guarantee that the packets would reach the destination. However, the IP made it possible to route packets from the source to the destination without setting up a circuit. The concept of relaying packets from source network to the destination network is known as *internetworking*, when both networks may use different technologies. *Routing* is what routers do, relaying the packets to the right destination.

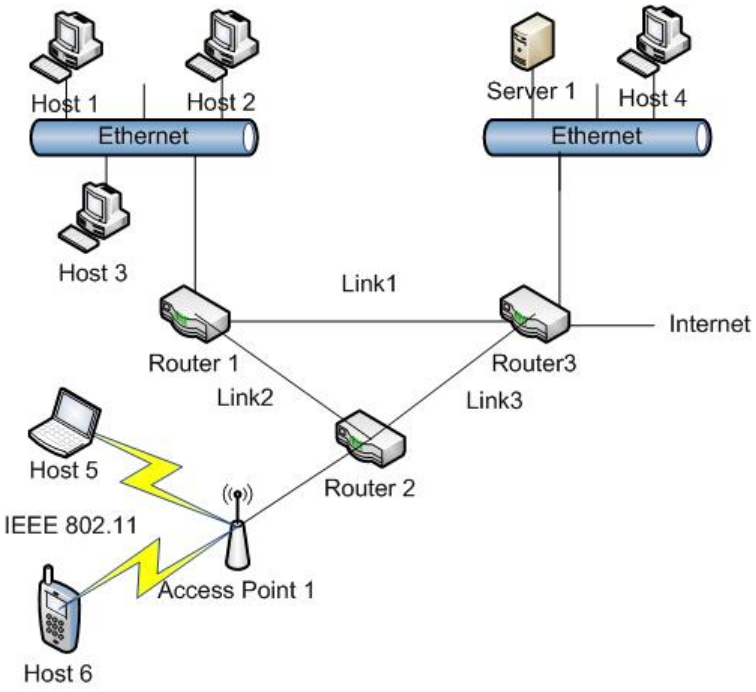

Figure 2.1: Simplified Internet infrastructure.

From the point of view of the network layer, the *Internet* consists of networks, routers, servers, and hosts. *A network* is a unit consisting of hosts, servers and a *router*, which does the packet relaying to the destinations outside the network. It is a special-purpose network element, which is used for relaying packets from one network to another. Routers provide connection for many different types of underlying network technologies as shown in Figure 2.1 [Ste96, page 4]. For example, Ethernet and IEEE 802.11 are different types of network technologies connected via routers. The physical medium connecting the underlying network technologies is known as a *link*. Hosts are organized to the networks managed by routers. *Hosts* are elements in the network that are capable of communicating using IP protocol. A *server* provides services for the hosts, which connect the server to use the services. Each individual host has an IP address or IP addresses so that it can be located in the network topology. The *topology* describes the organization of the hosts in the network. The routers relay the IP packets from the network to the other based on

the knowledge of the network topologies. A router manages the relaying of the IP packets for a certain designated address space.

IP addresses build a namespace for the Internet and every host has to have at least an IP address in order to communicate using IP. Depending on the IP version, an *IP address* is 32 (version 4) or 128 (version 6) bit long number sequence. Besides that, the Domain Name System [Moc83] builds the other principal namespace related to the current Internet. The *domain names*, structured of the Latin alphabet and Arabic numbers, establish a human readable namespace that provides names for the hosts. The *Domain Name System* is in fact a distributed database, which associates the domain names and the IP addresses. It stores the information in Resource Records (RRs), which are more precisely called A record and AAAA record in case of IP. Domain names are mapped to IP version 4 (IPv4) addresses in A records and to IP version 6 (IPv6) addresses in AAAA records. A host enquires an IP address of the known domain name or vice versa with the Domain Name System. [Ste96, page 9].

The Domain Name System is hierarchical. A *domain* consists of a set of network addresses. Each domain name composes a tree, which has a label for every domain. For example cs.tut.fi belongs to domain Finland (fi), beneath that Tampere University of Technology (tut) and then to the Computer Systems Department (cs). The Domain Name System consists of a hierarchical set of DNS servers, which are related to the corresponding domains. DNS servers publish information about the domain and all the servers under it in the tree.

2.1.1 Role and challenges of IP addresses

IP address has two roles: a topological information string and an identifier of a host. The IP address defines a point of attachment in the Internet topology. A host attaches the network via the *point of attachment*. The boundary between the host and the point of attachment to the link is called a *network interface* [Kur07, page 342]. This means that an IP address names the point of attachment instead of naming the respective host because the host may change in the current Internet. It describes the location of the point of attachment and after connection the location of the host in the topology. At the same time, it also acts as an identifier of the host [Mos06, page 11]. The IP address is all that a host has beside the possible domain name in the current Internet architecture. This means that the assigned IP address identifies the host. Figure 2.2 shows an example of these logical entities connected to an IP address. The dotted lines illustrate the roles, a host identifier and location information, associated with an IP address.

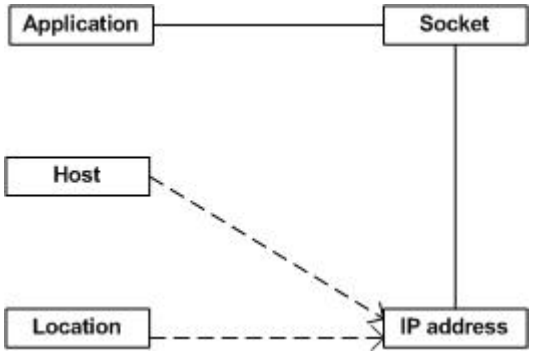

Figure 2.2: Logical entities connected to IP address.

Applications use IP addresses with the sockets. *Sockets* provide an interface for the application to access the TCP/IP stack. This has the consequence that IP addresses act in end-to-end manner binding the sockets to the location. This has many severe consequences. An application bound to a socket by an IP address is also bound to the topology. If a host changes the IP address, because of moving to another network, an application must break the upper layer bindings and restart the communication via the new IP address. This is very often the case with mobile hosts. When a host moves in the Internet topology from a network to another it is said to be *mobile*. The sockets associated to IP addresses, because a problem that must be currently solved in the application layer unless a technique called Mobile IP [C02] is not used Section 2.2.2. All the applications willing to work after *roaming*, moving from a network to another, without causing too much delay or disconnection, must implement a mechanism to handle the changing of the IP address associated with the socket. The overhead in applications causes extra implementation.

The Domain Name System (DNS) namespace binds a set of hosts or applications to a certain IP address so that applications can use DNS instead of using just IP addresses. Though domain names are in use, IP addresses still have a dual role as host identifiers and locators in the Internet architecture.

Besides the dual role of IP addresses, there are other deficiencies with current namespaces. First, semantic overloading and extended functionality complicate the IP namespace [Mos06, page 6] [Ste96, page 429]. Network Address Translators (NATs) divide the address namespace to the public and private namespaces that restricts identification and the validity of the topological information to the private address space. Second, the current IP namespace does not provide unique authentication of hosts or datagrams [Mos06, page 6]. In other words, current namespace is non-cryptographic; a host cannot be authenticated based on the IP address. Because authentication is lacking, reliable communication must occur by other means.

Third, the current namespaces does not provide anonymity of hosts [Mos06, page 6]. *Anonymity* refers to a situation where a host cannot be identified. There is no anonymity in the domain names. The consequence is that an unwanted party may be able to determine the communicating host.

2.1.2 IPv4 addressing

IP version 4 addressing is the base of the internetworking at the network layer. A unicast IPv4 address is a 32-bit digit consisting of a host part and a network part. *Unicast* address means an address that is used for one to one communication. This section concentrates on the unicast addresses. The host part refers to the single host in the network and the network part refers to the single network in Internet. The network part, also referred to as a *prefix*, defines the designated address space for a router to manage.

Prefixes are given based on the consideration of the authorities. The parent organization is Internet Assigned Numbers Authority (IANA), which allocates designated address spaces to regional Internet registries, who in turn allocate smaller designated address spaces to Internet Service Providers (ISP) and enterprises. In *subnetting* addresses are being divided into smaller networks called subnets [Ste96, page 42]. Subnetting is a technique to deal with arbitrary long prefixes. An address mask is used to define the prefix. The address mask consists of the significant ones. The number of significant ones defines the length of the prefix. The prefix is obtained by taking a bitwise AND operation of the IP address and the IP mask. A non-continuous prefix, which consists of non-continuous significant ones, is illegal.

Internet Protocol version 4 has limited address space due to the fact that the current address space is not fully utilized. First, an IP address is composed of a bit pattern and thus the network prefixes divide the address space according to binary arithmetics. For example, when in a single network there are 17 hosts, it takes 20 IP addresses, because in addition to the addresses of the hosts there must be at least a router, a network address and a broadcast address. The lower prefix allows 16 IP address and the upper 32. A prefix with 32 IP addresses must be chosen and that leaves 13 IP addresses unused. This is also known as a *scaling problem*. Second, the other fact is that Internet was designed for a smaller amount of hosts. An IP address was meant to be an identification for a single host but it is not possible to have an IP address as an identifier with the current IP namespace. The reason is the *scaling* problem, which has resulted in various techniques to support the increasing amount of hosts connected to the Internet. Techniques such as Network Address Translation (NAT) have prolonged the translation to Internet Protocol version 6.

2.1.3 IPv6 addressing

The Internet Protocol version 6 (IPv6) was introduced in the early 1990 by the Internet Engineering Task Force (IETF) [Kur07, page 360]. IPv6 introduces *scopes* that define the validity of a unicast address in the different zones. Link-local and global are currently the two different scopes defined for IPv6. The situation is illustrated in Figure 2.3. A link-local address is valid on the link it is attached to. A router delimits the link-local scope. In Figure 2.3 Router1 limits the link-local addresses that are valid only in zone 1. A global address is globally unique. Scopes add complexity to the routers, which have to deal with the validity of the source addresses before passing packets to the next hop.

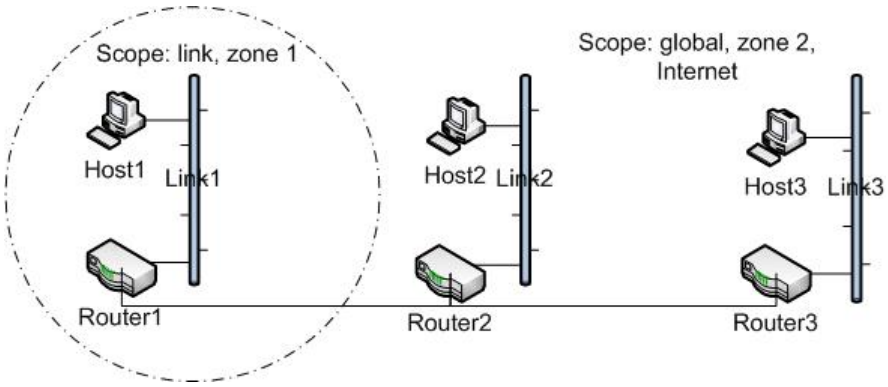

Figure 2.3: IPv6 address scopes.

A global IPv6 address is a 128 bits long address constructed of a prefix, subnet ID and interface identifier. A prefix refers to a global routing prefix, used to define the site the host belongs to [Sol04, pages 51-61]. The subnet ID indicates the subnet inside this site and the interface identifier uniquely defines an interface on a link. The subnet ID can be defined by a host itself on a link with a technique called stateless address autoconfiguration [Tho96] or it can be assigned. Stateless address autoconfiguration supports dynamic readdressing in a certain scope and makes it possible to a host to move to another network without any support from the network itself.

2.2 Host multihoming and host mobility

Multihoming as well as mobility exists in many contexts. *Site multihoming* refers to a situation where a site has more than one transit provider, which is an Autonomous System that offers transit of network traffic to or from some other AS. Site multihoming is out of the scope of this thesis, but multihoming exists also at the hosts.

Host mobility and host multihoming refers to the behavior of a host often using several networks in the parallel or one after another.

2.2.1 Host multihoming

End-host multihoming means that a host has more than one IP address assigned to it. There are two different cases of the situation. First, a host can have several points of attachment to the Internet. Second, a network interface can have several assigned IP addresses. End-host multihoming allows a host to use multiple IP addresses for sending and receiving IP packets. The situation where a host is multihomed introduces several challenges for communication to host itself, for the peer and for the network.

A host that has several points of attachment to the Internet can choose different paths to a peer. Often points of attachment are attached to the different links. Different links offer paths that can have different bandwidths, transfer delays, fault tolerances, or costs. Load balancing is one reason to utilize several paths at the same time. *Load balancing* refers to a situation where traffic is spread between multiple network paths in order to achieve better throughput or response time. If the changes in the path are not taken into account by the host, the upper layer bindings are disconnected and the application cannot communicate through the same socket anymore. This connectivity change is often referred to as a *handover*. It may stop the communication between peers or it may delay the data transfer. The other end of the communication has to be aware of the handover to be able to send the data to the right destination. To seamlessly support these scenarios host multihoming must be supported at the both ends of the communication.

On the other hand, the existing network architecture must support different host multihoming scenarios. A host must have multiple DNS AAAA or A records enabled to be reachable in case of having multiple IP addresses. After an IP address becomes unusable, the communicating peer can connect the host via, a new IP address received from a DNS query. This requires that the peer announces its IP addresses for the DNS often enough. On the other hand, the used protocol could have the mechanism inbuilt so that the communicating peer knows already beforehand what the new address will be. The latter approach does not work if the connection is lost accidentally without having the possibility to inform the peer. If the connection is lost accidentally without having the possibility to inform the peer, it is known as a *hard handover*.

2.2.2 Host mobility

Host mobility is a phenomenon covering the mobile host movements from a network to another. Host mobility and multihoming are closely related together because a host changing its point of attachment needs to obtain a new IP address in many cases. A mobile host may be *single-homed*, i.e. it has a single IP address assigned, but it is capable of changing the IP address. The situation is similar to host multihoming when a host changes the path for communication with the peer. To fully support mobility, the network has to provide a way for the communicating peer to continue the communication with the relocated host. There are a couple of approaches to solve the issue.

IETF introduced a technique called Mobile IP [Per96] already a decade ago to solve host mobility. A Mobile IP node, i.e. a host, is given a long-term IP address on a home network. This IP address, which is referred to as a home address, is used as a source address of all IP datagrams that it sends except certain mobility management cases. When the Mobile IP node moves in the network topology, a Care-of-Address is assigned for it in the foreign network. It reflects the current point of attachment of the mobile node. The Mobile IP system consists of a Home Agent and a Foreign Agent that handle the data flow between home address and the Care-of-Address. The system provides for registering the Care-of-Address with a Home Agent. The Home Agent sends datagrams destined for the mobile node through a tunnel to the Care-of-Address. After arriving at the end of the tunnel, the Foreign Agent detunnels and delivers the datagrams to the mobile node. That is the basis of the Mobile IP architecture. [Per96]

The Mobile IP solves the host mobility in the macro scale. In the micro scale, the Mobile IP does not provide the best solution. If the handover happens between points of attachment in the same subnetwork, the Mobile IP mechanism has a lot of overhead [Per96, page 4-5]. Besides that, HIP has been introduced as one solution to support both host mobility and host multihoming.

2.3 Security architecture for the Internet Protocol

The security architecture for the Internet Protocol is known as IPsec, and it is based on usage of cryptography. IPsec is closely bound to HIP and IP protocols. Two HIP hosts are typically, but not necessarily, protected with IPsec [Mos06, page 2].

2.3.1 Introduction to cryptography

Integrity and confidentiality are essential concepts in the field of data security. *Integrity* defines that data is not changed or destroyed over a time interval. Integrity is often related to data transfer tested by the receiver of a message, the data. *Confidentiality* is a higher-level concept and defines the rights to access the data. Authentication, encryption, and decryption are methods to cover the confidentiality. Assurance of the identity of a party is called *authentication* and assurance of the rights of a party to access data is called *authorization*. *Encryption* is a term for making some data secret and *decryption* is a term for making the secret data clear again. The secret data is often referred as ciphertext and the clear data as plaintext. [Kos06]

Cryptographical functions have in most cases a secret and an algorithm. This is a common trend for cryptographical functions [Kau95, page 40]. The secret is often referred as a *key*. The output of the cryptographical function with chosen algorithm has *computational difficulty*. This means that the output of the well-known algorithm cannot be solved back to the input without reasonable amount of effort. That is often enough and nothing more is even expected. Also *Perfect secrecy* can be achieved. When the ciphertext is under attack and it does not reveal any information about the plaintext, the system is perfectly secure.

Cryptographical functions can be divided in to three parts: *hash functions*, *secret key functions* and *public key functions* [Kau95, page 45]. Hash function does a transformation from an arbitrary long message to a fixed-length message. This is a one-way transformation and the output cannot normally be transformed back to the input. Special *keyed hash-function* is used to produce digest for authentication. Message Authentication Code (MAC) is one type of keyed hash-function. A key is added in the end of the input and the new input is hashed. Produced digest is added to the message and reproducing it, the receiver must know the related key. It is a method to authenticate the data origin. IPsec and HIP both uses special version of MAC known as HMAC. It is keyed hash inside a keyed hash. In other words, the hashing is done twice by adding the result of the first round to the second. HMAC uses also some padding to produce the digest [Dor99, page 14-15].

When a message is encoded and decoded with the same key, the responsible cryptographical function is symmetric also known as secret key function. On the other hand, if one uses different key for encryption and decryption the cryptographical system is asymmetric. *Public Key Cryptography* (PKC) is asymmetric cryptographical system and was invented in 1975 [Kau95, page 48]. These asymmetric functions in PKC are known as public key functions.

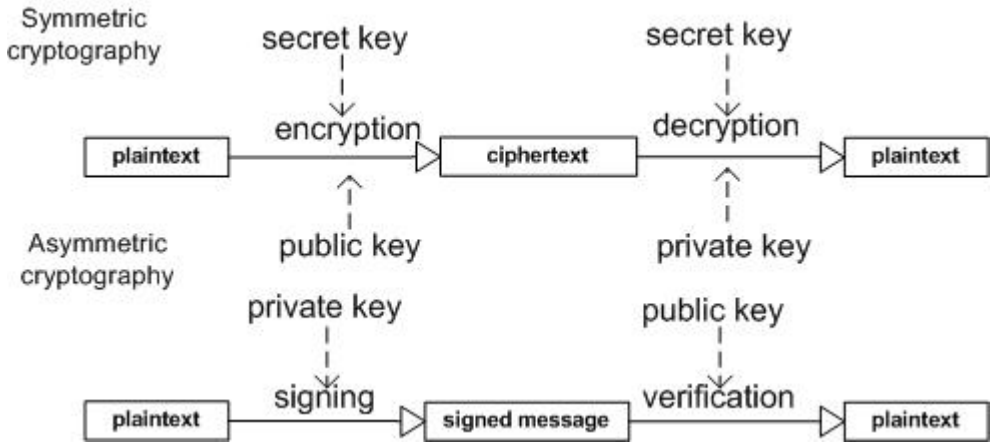

Figure 2.4: Symmetric and asymmetric cryptographical functions.

In PKC, interacting party has a private key and a public key. The private key is meant to be known only by the associated party. The public key is provided for all parties interested in it. Private key that is needed for decryption and signing of a message should be known only by the trusted parties. *Digital signatures* are important application of PKC. Authentication of the messages can be based on these by attaching the digital signature to the message before sending and then verifying it on the receiver side. Digital signature can only be generated by someone knowing the private key and it can be verified by anyone knowing the public key. Because private key should be known only by the sending party, it cannot be repeated unless someone knows it too. With a secret key function, it is not clear which one of the parties possessing the secret key created the signature. This has an implication that the signature created with the secret key function is ambiguous. It can be *repudiated*. Digital signatures generated with PKC fulfill the non-repudiation demand and because of that are very important inventions. Digital signatures are more reliable than MACs because no one should be able to forge them. In the other hand, the algorithm used to produce the signature is often costly and that is why MACs are preferred. In PKC encryption and decryption works other way around compared to signing. Encryption can be made by anyone possessing the public key but for the decryption needs private key of the key pair. [Kau95, pages 48-53]

2.3.2 Overview of the IPsec

IPsec, Security Architecture for the Internet Protocol, is designed to provide security at the Network layer. IPsec provides security with Authentication Header (AH) and Encapsulating Security Payload (ESP) for data communications hidden from the

layers above that was one of the main objectives for the design. Existing applications do not need to be changed and security unaware users can benefit from the security features without paying attention to the encryption and authentication of the data communication. These arguments were presented to support the approval of the design. [Tan03, page 772].

IPsec implementation must have two crucial parts, the key negotiation protocol, and the actual cryptographical services. Cryptographical services are based on the existing network architecture and cryptographical functions. IPsec is by default used with key negotiation protocol called Internet Key Exchange (IKE) protocol version 2 [Ken05b, page 47] to establish the shared secret keying material and Security Associations (SAs). In order to secure traffic with IPsec the security policies (SPs) for system must be defined. Both SAs and SPs have their own databases to store them. These are called Security Association Database (SAD) and Security Policy Database (SPD). This work concentrates on the above-mentioned concepts, the actual services, and a key negotiation solution known as HIP. IPsec is a central part of HIP that uses IPsec to carry encrypted data traffic and to support the logical separation of end-points and IP addresses by using the new introduced IPsec mode BEET. HIP consist of several features in addition to the key negotiation, which are discussed in Section 2.4.

IPsec services AH and ESP

IPsec provides services for authentication and encryption of data traffic. Authentication Header (AH), which is added right after IP header, provides integrity, data origin authentication, and an anti-replay service [Ken05a, page 2]. The authentication and integrity checks are by the default done with keyed hash functions HMAC-MD5-96 and HMAC-SHA-96 but other functions can be used too. Digital signatures such as RSA and DSS are not used due to the cost of the operations. [Dor99, page 92]. RSA and DSS consume too much CPU resources and time. Anti-replay service is implemented by sequence numbers, which are increased by the sender and ensured by the receiver. This protects against certain kind of attacks where a third party tries to replay a packet. Denial-of-Service (DoS) attack tries to degrade the availability of services available at network by causing resource exhausting computation for the receiver. This anti-replay service protects against these by allowing receiver to do a simple check for the received packet and discard it if the sequence number is not correct. AH is mostly used for the upper layer protocols to support the data origin reasoning. [Ken05b, page 76]

Encryption service is called Encapsulation Security Payload (ESP) and it is a

header added right after the IP header like AH. ESP is used for confidentiality, and limited traffic flow confidentiality. Confidentiality is provided with encryption service. Traffic flow confidentiality is implemented with IPsec tunnel mode by encrypting also the original IP address of the sender. In addition to these, ESP provides connectionless integrity, data origin authentication, and an anti-replay service. [Ken98, page 6-7]. In fact, ESP provides the same services as AH in limited form. ESP implements authentication in the same way as AH with the chosen algorithm but it does not include the IP header to the calculation of the digest. In that sense, it is more limited and suitable mostly handling of upper layer Protocol Data Units (PDUs) [Ken98, page 10]. It actually ensures the integrity of the PDU. The digest is added to the end of the IP packet as a separate field, Integrity Check value (ICV). There are several possible algorithms that can be used. It is noted that IPsec must implement following algorithms: DES-CBC for encryption and HMAC-MD5-96 and HMAC-SHA-96. The keying material for the services is derived from manual keying or by the result of some key agreement method.

IPsec has two modes: a transport mode and a tunnel mode, as depicted in Figure 2.5). Both services AH and ESP supports these. Transport mode is meant for end-to-end security. Transport mode means that the IP packet will have extra fields containing the information related to the AH or ESP. In tunnel mode, a new IP header is constructed in the front of the packet and right after that is placed the constructed IPsec specific header. Tunnel is constructed as IP-IP tunnel [Sim95] but everything after the first IP header is applied to the wanted IPsec security services. Tunnel mode is meant to be used with Security Gateways.

Security Policies and Associations

In order to communicate properly IPsec must establish correctly Security Associations. Security Association is logical channel between the network layers of source and destination hosts [Kur07, page 729]. There are two types of SAs, unidirectional and bidirectional. In IPsec, unidirectional SAs are used so we concentrate on those [Dor99, page 44]. A pair of Security Associations is installed. An SA must be installed for both directions of the data transfer: one for the outgoing traffic and one for the incoming traffic. Altogether two communicating parties using IPsec need four SAs for that. SA has an identifier related to it; Security Parameter Index (SPI) that is carried also by the actual IPsec secured traffic. The SPI is used by the receiver to identify the incoming traffic. It is a unique identifier and supports that way multiple simultaneous SAs between the same hosts. SA includes information of both ends: address and possible identity. SA relates the needed algorithms to

the connection and has a lifetime. The lifecycle of an SA has four stages: Larval, Mature, Dying, or Dead. This stage defines when the SA can or cannot be used between two parties.

Security Policies defines how and what of the real traffic of the respective host should be secured. SP is a description consisting of information, defined by the (*selectors*), which is wanted to be secured. Selectors are typically IP address, port, or protocol-specific. SP has also other task. SPs relate the traffic to certain IPsec actions: application of AH, ESP or tunnel [Dor99, page 131-133]. Instead of having some agreement method for policies management, interface and application must be defined. Policy definition is human-computer action and thus along IPsec, a manager using an interface for these must be defined to be able to define wanted IPsec transforms. *Transforms* is a term for the methods to enforce the wanted IPsec actions.

2.3.3 A Bound End-to-End Tunnel

IPsec provides two modes for the services AH and ESP to carry the data traffic, tunnel and transport mode. A new mode called a Bound End-to-End Tunnel mode (BEET) has been defined in a recent IETF draft [Nik07b]. It provides tunnel mode semantics without tunnel mode overhead. Packet format in the wire is like in transport mode (Figure 2.5) but the semantics of the connection using BEET are different. Modes are similar in IP version 4 and 6.

Tunnel mode implements IP-IP tunneling used with the Security Gateways. When ESP is in use, inner IP header is encrypted. This scheme implements encrypted tunnel between Security Gateways. Tunnel mode support IPsec tunnel use with Virtual Private Network (VPN) but not in end-to-end use. BEET implements the concept of Security Gateways at the end hosts themselves. This is done with a pair of addresses defined in SAs called inner and outer address. On the outbound side, processing before ESP uses the inner addresses and after that, a new header including the outer IP addresses is constructed to replace the original IP header. When a piece of more recent design, HIP, is used, this provides so-called *channel binding* [Hen07b, 3]. In case of HIP, the inner addresses are HITs providing cryptographical channel binding with public keys, HITs (Section 2.4.1).

There are several use cases for such a binding channel. In order to support such features as NAT traversal and mobility and multi-homing, some extra features must be implemented with BEET. The transformation of inner address to outer addresses is done by the Security Association of IPsec, which holds the inner address as an identity of the host. Applications bound to these identities could be capable of

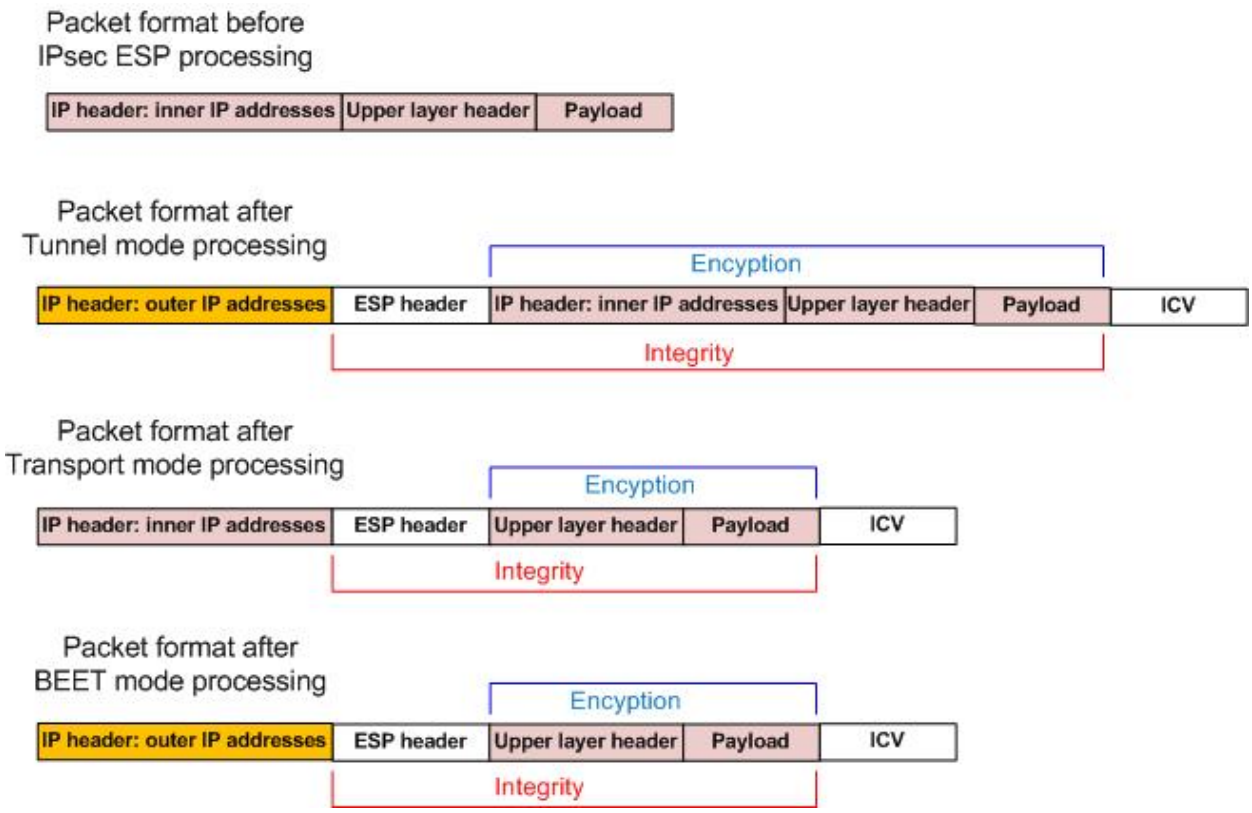

Figure 2.5: IPsec traffic modes and new mode BEET.

changing the outer address and even the upper layer bindings stays connected. In this case changing the IP address would allow maintaining the connection to the peer host while communicating. As a part of HIP, BEET provides these facilities at some level but also several other things must be taken into account in the implementation. The implementation aspects of BEET are discussed in Chapter 5.

2.4 Host Identity Protocol

Host Identity Protocol (HIP) is a joint effort of IETF HIP Working Group members (WG). Even though Host Identity Protocol (HIP) is an end-to-end authentication and key establishment protocol used with IPsec ESP [Mos07, page 6], it is designed to support host mobility and end-host multihoming. The details of HIP PDU format and state machine are introduced in Chapter 4.

2.4.1 A new namespace

To fix the deficiencies with the current namespaces, the IP address namespace and the Domain Name namespace, HIP introduces a new Host Identity (HI) namespace.

The Host Identities are the building blocks of the new namespace. The new namespace forms the upper layer bindings to the Host Identities. This splits the duality of the IP addresses because the identity of a host is related to the HI instead of the IP address. A Host Identity separates the identity of the host from the location information that the IP address carries. Figure 2.6 introduces the situation [Mos06, pages 2-11]. The new namespace fills the gap between IP addresses and DNS names by releasing IP addresses from the upper layer bindings. Besides that, by introducing the new namespace the semantic overloading and extended functionality of the current namespaces could be avoided. For example, HIP has a proposal to solve NAT traversal.

Hosts are still located with IP addresses but the identity of them is bound to the Host Identity as shown in Figure 2.6. Sockets are bound to the Host Identities. This provides solution for the roaming problem: a host can move from network to another using same host ID and maintaining the sockets connected. A Host Identity does not change if the location changes. The new namespace provides thus for a host a possibility to be reachable via several IP addresses, because the IP address can be changed seamlessly [Mos06, page 11].

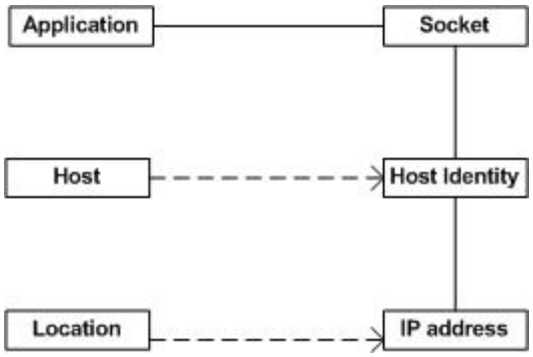

Figure 2.6: Logical entities connected to IP address along new Host Identity.

A HI is a public key of an asymmetric key pair. Because it is cryptographic in its nature, it tries to tackle the problem with the authentication challenges of the current namespace: the IP address does not provide any means to authenticate the peer. A host may have a published and an unpublished Host Identity. In both cases, the peer authenticates the HIP PDUs with the Host Identity, it has received with the DNS query or directly from the peer. However, there are minor vulnerabilities with the new namespace (see Section 2.4.5).

Anonymity can be achieved via unpublished locally created HIs that are not registered [Mos06, page 7]. In this manner, the unpublished Host Identity stays anonymous and the privacy can be achieved. When privacy is achieved, security

improves. An unwanted party cannot determine the identity without help of the respective host.

HIs and HITs

In practice, a Host Identifier is introduced for every Host Identity. It is a unique bit sequence representing the public key but it is not used in the real communication. A truncated 128 bits long form called Host Identity Tag is formed for the real communications. It is formed by taking a cryptographical hash over the Host Identifier. The length is chosen the IPv6 socket compatibility in mind and 32 bits long local presentation called Local Scope Identifier (LSI) is introduced to support the deployment with IPv4 sockets in the beginning. [Mos06, pages 8-10]. HIT is also *self-certifying*, i.e. it is computationally hard to find a HI that produces matching HIT [Mos07, page 10] and that is why the probability of HIT collision is very low. There are also other reasons to have fixed length HITs. Fixed length tag represents Host Identity in consistent way. The used algorithm for public key infrastructure it relays on does not make difference to the length in case of HITs. Another reason for fixed length is easier protocol implementation compared to variable length identifier. [Mos06, page 10].

Host may have several Host Identifiers to identify itself. Some of the identifiers are public and some of them are unpublished. A public identifier could be stored to DNS server and peer willing to communicate with a certain host can fetch the HI by DNS query and try to authenticate itself for the communications. A Public Host Identifier and the related HIT is published in DNS record and unpublished HI is known only by the host itself, and the peer for whom it was published. DNS extension is one possibility to the basic architecture. In addition, other proposals have been made.

Change to layer architecture in TCP/IP stack

Host Identity layer changes the Internet stack architecture introducing a new layer between network and transport layers. In contrast to earlier architecture transport layer handles Host Identifiers instead of IP addresses. HIP layer manages these bindings. HIP layer maps IP addresses to Host Identifiers.

The new layer may be implemented using the IPsec. The new IPsec mode BEET holds both IP address and Host Identifier in a Security Association and changes the used namespace between the ESP processing. This situation was illustrated with the IPsec BEET mode in Section 2.3.3. The outer IP address presents the real IP address of a host and the inner IP address is actually a Host Identifier Tag in IPv6

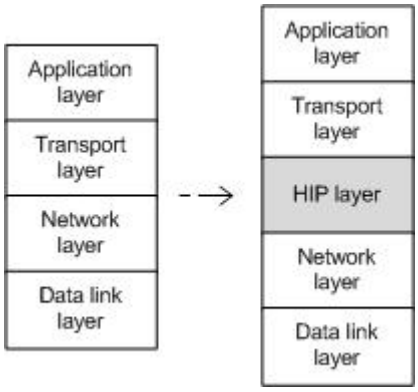

Figure 2.7: New stack architecture.

and Local Scope Identifier in IPv4.

2.4.2 HIP Base Exchange

The HIP Base Exchange is based on the Diffie-Hellman key agreement method. It provides a way to establish shared secret, which is used for encryption of the HIP PDUs.

Diffie-Hellman key agreement

Diffie-Hellman (DH) key exchange [Dif76] is a public cryptographical system to share a secret between parties. Using Diffie-Hellman key exchange, a public communication channel can be shared by the parties in order to communicate securely. Diffe-Hellman is only a public key distribution system and it does not offer anything else than that. Authentication of the messages must be handled separately. HIP Base Exchange adds authentication in addition to other things to the DH key agreement.

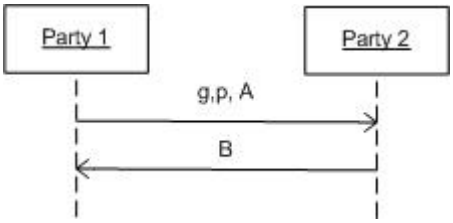

Figure 2.8: Diffie-Hellman key exchange.

Diffie-Hellman key exchange is illustrated in Figure 2.8. DH key exchange uses two messages to establish the shared secret as shown in Figure 2.8. First, the two

parties 1 and 2 agree on the group: prime p and generator g. The group is the base of the algorithm. Then both parties have to choose a random number to calculate a public number. Party 1 chooses a and party 2 chooses b. The calculation of the public number is done as follows,

$$A = g^a \bmod p$$

$$B = g^b \bmod p$$

The generator is exponentiated by the random number (a or b) and module p is taken from that, to calculate the public number [Dor99, page 16]. The first message consists of the chosen group (g,p) and the public number (A) of party 1. After this party 2 sends only its own public number (B), and shared secret can be established.

The following formula invented by Diffie and Hellman [Kau95, page 148] calculates the shared secret K, which is also used as a shared cryptographic key,

$$\begin{aligned} K &= B^a \bmod p \\ &= (g^b \bmod p)^a \bmod p \\ &= g^{ab} \bmod p \\ &= (g^a \bmod p)^b \bmod p \\ &= A^b \bmod p \end{aligned}$$

When both parties know p,g, the individually chosen random number, and the public of the other, the shared secret K is straightforward to calculate. By exponentiating the public of the other with own random number and taking modulo p of that, gives the shared secret K.

Base Exchange protocol

Initiation of HIP connection is based on HIP Base Exchange (BEX) and it is performed before connection is established. When the HIP connection is established, a HIP association to both directions between two peers is created. BEX is a one-to-one four-packet exchange protocol based on Diffie-Hellman key exchange (Section 2.4.2). It imbeds following parameters in addition to Diffie-Hellman variants to the PDUs: HITs, a counter, HIP_transforms, HOST_ID -parameter, echo_request / echo_response and a signature as illustrated in Figure 2.9. Second and third PDUs are used to establish the shared secret and third and fourth PDUs are used to do the authentication. The HIP architecture document [Mos06] or Base Exchange draft [Mos07] does not define any transport format to use in HIP communications but it is mentioned that IPsec ESP transport mode must be used at minimum for

Encapsulating Security Payload. ESP BEET mode (Section 2.3.3) has many advantages compared to ESP transport mode so that is preferable. Figure 2.9 shows the Base Exchange PDU by PDU.

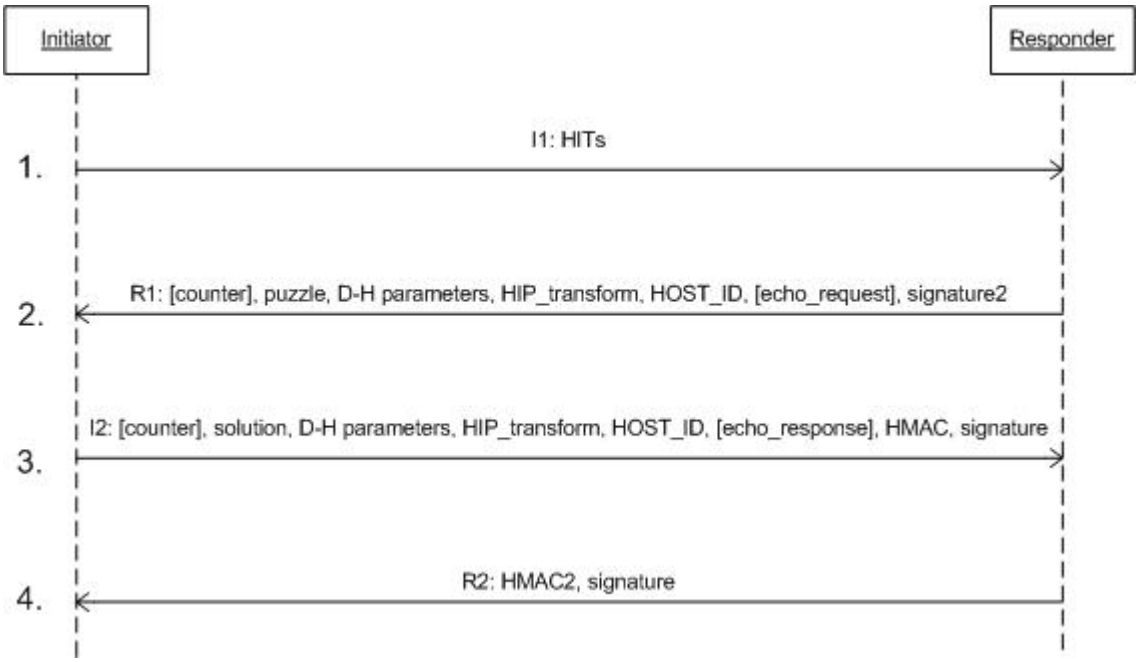

Figure 2.9: Host Identity Protocol Base Exchange.

Initiator sends the first rather simple message, I1, which triggers BEX. It consist of the corresponding HIP header, the Host Identity Tag of the initiator and the Host Identity Tag of Responder, if it is known [Mos07, page 12]. If Responder's HIT is unknown, Initiator may fill the fields with zeros. This kind connection setup is called opportunistic mode [Lin06]. In some cases it may be possible to replace this trigger packet by some other form of a trigger but these cases are undefined [Mos07, page 12]. In such a case, the protocol starts with Responder sending the R1 packet. Connection state is at this point UNASSOCIATED (state diagram is shown in Section 4.1).

Responder may refuse to continue BEX due to malformed message, local policy decision or if HIP is not supported. It may decide to respond with appropriate ICMP message 'Destination Protocol Unreachable' if it does not support the protocol or if the local policy denies this HIP connection ICMP 'Destination Unreachable, Administratively Prohibited' message may be sent. If I1 was successfully received, it generates the R1 message.

The second message, R1, starts the actual key exchange. It consists of the multiple Diffie-Hellman (D-H) parameters. A D-H parameter consists of group id, p and

g that defines the used public Diffie-Hellman key. There can be at the maximum two group ids. The idea behind having more than one group id is that the devices with limited sources of power can choose weaker encryption but in the normal case, Initiator selects stronger encryption [Mos07, page 44]. Second parameter is a puzzle, which Initiator tries to solve. Puzzle is based on a hash algorithm over HITs, IP addresses and a random number I. The difficulty of a puzzle is set by number K. Initiator uses a hash algorithm for computation of the puzzle. Responder sets a difficulty based on the level of trust, the current load, or other factors [Mos07, page 13]. Counter indicates the freshness of the message. *Freshness* indicates how recently the message has been sent. HIP_transform consist of the authentication and the encryption algorithms supported by Responder. It is a list of items where the preferred algorithms are in the beginning of the list. HOST_ID parameter contains a Fully Qualified Domain Name (FQND)/Network Access Identifier (NAI) and a Host Identifier. Signature2 is a digital signature attached for the authentication of the message. Number two indicates the version. The digital signature is calculated with the private key corresponding to the Host Identity of the corresponding party [Mos07, page 74]. R1 message is authenticated by verifying the signature of the responder by using the HI as a key. To continue the negotiation Initiator needs to solve the puzzle correctly. Counter is monotonically increased by a random 64-bit number. The proposed D-H parameters may be refused. In order to successfully continue BEX, one of the proposed group ids is selected. Initiator chooses an authentication and an encryption algorithm from the HIP_transform list and replaces the choices in the parameter. Echo_request is copied to I2 message and echoed back in echo_respond parameter. This is optional feature.

The third message, I2, has a solution for the puzzle. If the solution is not correct, the packet is dropped. Counter is used for the same purpose as in R2 message: checking the validity and the freshness of the puzzle. D-H group choice is sent in the message. Initiator chooses the most suitable cryptographical algorithms and returns the information of the choices in HIP_transform parameter. Last are added HOST_ID, echo_response, HMAC and Signature of Initiator. HOST_ID may be encrypted with the keying material derived from the Diffie-Helman key exchange.

This is the point of ending the Diffie-Hellman key exchange and the state creation at Responder side. State creation is delayed until verification of I2 message (see Section 2.4.5). Responder compares the counter to the expected one and finds out the freshness of the solution before ensuring the correctness of it. Correctness of sender's HIT is also studied with the signature in the same manner as earlier. After successful verification of preceding parameters, D-H choice and HIP_tranforms, HIP

association is installed at the side of Initiator.

Fourth and last message, R2, is simpler than messages R1 and I2. It is the last message before the connection transfers to ESTABLISHED (state diagram is shown in Section 4.1). It is a signed response that informs the Initiator of successful negotiation. Both HMAC2 and a HI based signature are calculated over the entire PDU. HMAC2 is the second version of HMAC.

2.4.3 Using ESP with HIP

In order to use IPsec Encapsulation Security Payload some minor changes are needed to the protocol as shown in Figure 2.10. The changes are defined in a HIP extension [Jok06]. These changes are written ESP transport mode in mind but everything what is presented here, is applicable for the ESP BEET mode as well.

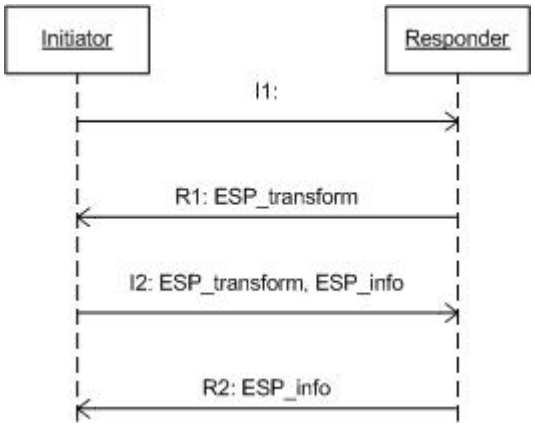

Figure 2.10: Setting up an ESP SA between HIP hosts during BEX

ESP_transfrom consist of authentication and encryption algorithms supported by Responder in R1 -PDU. Initiator chooses the used suite and gives the response in I2 -PDU. The transform suite IDs is carried by the parameter has following options for authentication and encryption:

- ESP-AES-CBC with HMAC-SHA1 with suite ID 1
- ESP-3DES-CBC with HMAC-SHA1 with suite ID 2
- ESP-3DES-CBC with HMAC-MD5 with suite ID 3
- ESP-BLOWFISH-CBC with HMAC-SHA1 with suite ID 4
- ESP-NULL with HMAC-SHA1 with suite ID 5
- ESP-NULL with HMAC-MD5 with suite ID 6

ESP_info consist of the SPI value that indicates the correct SA, which is used for the ESP traffic between two peers in question. In I2 -PDU Initiator returns the SPI value it uses for the incoming ESP traffic to indicate the correct SA. The Responder will install an outgoing SA based on that same SPI it got from ESP_info. Finally, the Responder installs the incoming SA and sends the corresponding SPI value along R2 -PDU in ESP_info parameter to the Initiator. Initiator installs an SA based on this SPI value for incoming ESP traffic. This procedure makes it possible to choose different SPI values for each HIP peer. [Jok06, pages 7-8]

2.4.4 Host mobility and multihoming with HIP

First, HIP introduces a new address space, which is closely presented in Section 2.4.1. This new addressing realm separates IP addresses from the above layers and forms a new layer in between. This has important consequence to the end-host multihoming and host mobility management in the HIP protocol. Because IP addresses are not anymore used by the applications or by the transport layer, changing of IP addresses becomes conceptually more consistent with the layered stack architecture. Applications do not need to take care of IP addresses, because it is done with the processing of IP packets in the HIP layer automatically. Then, host mobility can be supported with DNS. A new DNS HIP record, which is another RR record, is defined and used for solving the HIs to IP addresses and vice versa. HIP record maps domain name to the Host Identity. The HIP RR records both HI and the HIT. Hosts in different domains can enquiry the HI and the related HIT of the peer with DNS. Afterwards, the host could start the HIP Base Exchange. [Nik07a]

HIP mobility supports IP address changes at the both ends of the connection even if the communication is already ongoing. When a hard handover happens, it is possible to start the communication by using another path and establish the connection again with HIP UPDATEs. A HIP host can choose to accept a HIP packet from the unknown source address if it is integrity protected with the IPsec, but may have some implications to security (Section 2.4.5). Thus, HIP host sends a HIP READDRESS packet and the peer checks the reachability of the HIP host based on the IP address it got within the HIP READDRESS packet. Another thing is that, HIP uses ESP to receive packets from any address using HIP created SA [Hen07a, page 9]. This has some implications to the implementations (presented in Section 5.2). HIP is closely integrated with IPsec, which is thought to be advantage. However, this approach has disadvantages. The deployment of IPsec is needed to make HIP usable [Hen03, page 26]. In addition, IPsec adds high cost in terms of performance because of the heavy handshake procedure it adds to the HIP protocol.

An implementation for light-weight key establishment already exists [Hee06]. One advantageous point in HIP is that it adds no overhead to the protocol packets other than what IPsec does.

2.4.5 Security considerations

The new Host Identity namespace increases the security of the current Internet architecture. Host Identity Protocol authenticates the packets, which peer sends with Public Key Infrastructure based on the Host Identities, which is the basis of the protocol authentication mechanims. That decreases the possibility to Man-In-The-Middle (MITM) attacks. *Denial-of-Service (Dos) attacks* are based on the resource exhausting computation. HIP defends these by delayed state transition. The first response packet does not have computational effort. The computation, solving the puzzle, is done at the initiator side within the Initiator 2 packet. Responder has pre-computed Responder 1 packets and one of those is chosen based on the Initiators 1 packet. This prevents the computation effort and spoofing of R2 packets [Mos07, page 14]. HIP has still known vulnerabilities. However, MITM attacks are difficult completely to defend against. That would need third party authentication such as DNSSEC and denial of unpublished HI usage to make the Host Identity theft difficult. MITM can use public HI or unpublished HI to attack against a HIP party. Another known possible MITM attack relates to responders ICMP 'Destination Unreachable, Administratively Prohibited' message. Initiator should react to these after reasonable amount of time if appropriate HIP packet is not received. In other case, the response is easy to spoof and DoS attack is possible. ICMPv6 'Parameter Problem, Unrecognized Next Header' and ICMPv4 'Destination Unreachable, Protocol Unreachable' messages can be used to execute DoS attack against the Initiator. The protection against this is similar as in case of other ICMP message. The Initiator should not take any action to the ICMP message before appropriate time interval is spent. In case of multihoming and mobility, unknown source IP addresses could be supported. The reachability of the unknown IP address must be checked before sending any large amount of data to prevent possible DoS flooding attack against third party. An attacker could open many high-volume HIP connections with number of hosts and inform them that it has moved to the target IP address. If hosts blindly accepts this, a DoS flooding attack is ready for the target IP address. [Mos06, page 13]

2.5 Summary

The current namespaces, IP and Domain Name, provide challenges for the current Internet, which is not designed taking into account current demands. The current Internet has many mobile hosts and the number of hosts increase all the time. Because of that, host mobility and multihoming have to be solved. HIP protocol provides one solution to improve the current Internet Architecture. It supports host mobility and multihoming. That is important point of view when hosts are mobile and roam between IP networks. Besides that, HIP introduces a key negotiation protocol, which relates closely to the IPsec. The IPsec provides possibility to establish secure communication channel between peers. The architecture of the solution is well-defined, and thus it is ready for the deployment.

3. SYMBIAN OPERATING SYSTEM

This chapter describes the Symbian OS version 9.2. Especially, the Symbian-specific features related to the network software and the protocols are discussed and compared with the same parts in Linux, which is another popular OS for mobile devices. First, this Chapter introduces Symbian OS essentials related to the implementation. Second, this chapter studies networking architecture of the Symbian OS. Finally, the chapter summarizes covered topics. Symbian networking architecture is also studied in Section 5.3.2 and in Chapter 6.

3.1 Symbian OS overview

The Symbian Operating System is a 32-bit multi-tasking OS. The Symbian kernel handles only a minimum set of operations, hence it is a *micro-kernel* [Tas00, page 255]. *Kernel* is the core of the OS and it provides the lowest abstraction layer for the resources. The Symbian OS provides most of the functionality via a set of libraries included into the platform itself. *User-space* is the memory area where all unprivileged applications runs. On the contrary, *kernel-space* is strictly reserved for running the kernel, the privileged software of the OS. The Symbian platform consists of subsystems, which provides functionality for the user-space implementations. Each subsystem has a set of libraries providing particular functionality and the Application Programming Interface (API) to access it. *API* is well-defined software interface, which makes the usage of the software a standard. All Symbian APIs are not public, because the platform security restricts the access to the internal developers and the licensed partners (see Section 3.1.2). The *Platform security* controls the access and the execution of the binaries. Figure 3.1 introduces the subsystems of Symbian OS 9.2. The most relevant of them are underlined.

The subsystem Base contains the essentials of the Symbian environment: the basic types, the memory and system resource management and the structures for handling the concurrency. The emulator, which emulates the Symbian device, is part of the Base. The most essentials of these are introduced in Section 3.1.1. Basically, the Base includes everything that is needed for basic programming without User Interface (UI) or communications with other entities. For HIP implementation,

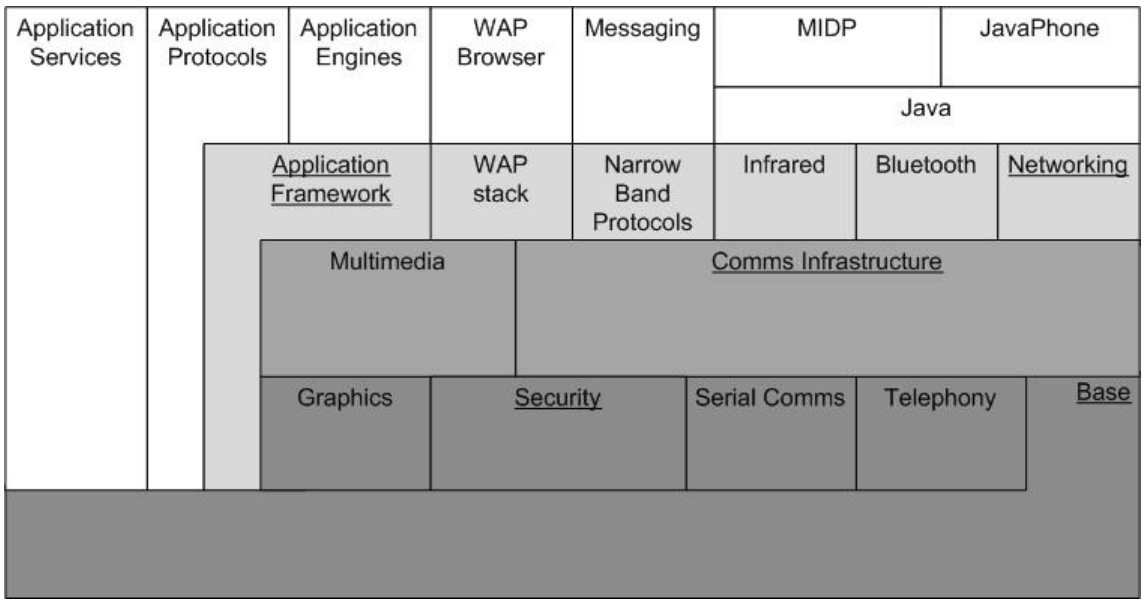

Figure 3.1: Symbian OS 9.2 is composed of several subsystems. [Sym06]

there are also other important subsystems: Comms Infrastructure, Networking, Security, and Application Framework. First, Comms Infrastructure contains the socket server, ESock (Section 3.2.1), to support socket API usage. Socket API is used for inter-process communications in the HIP protocol as well as a socket interface for HIP control messages. The Networking Subsystem holds protocol implementations and APIs related to them. HIP uses TCP/IP stack and IPsec in the Networking subsystem. IPsec and HIP need cryptographical algorithms. Cryptographical algorithms can be found from the Security subsystem. Applications framework includes a software framework for implementing User Interface for applications. *Framework* is a set of classes that embodies an abstract design for solutions to a family of related problems. In other words, a framework is a partial design and implementation for an application in a given problem domain [Bos97, page 3]. Application protocols define HTTP APIs. Messaging subsystem has support to build messaging clients such as POP3 and SMTP Internet mail, SMS, and fax. For calling services, there is telephony subsystem [Sym06]. Other subsystems are irrelevant for the thesis.

Symbian implementation is based on the existing modular software, the subsystems, that are provided for the foundation of the programming. Applications, and the components integrated to the platform have much interaction by other software modules. Thus programming in Symbian needs understanding how such a foundation and interaction are used. The native Symbian programming is based on C++ programming language. All the APIs, design patterns, and software frameworks provided for programmer are implemented with C++ extended with special Sym-

bian coding conventions. Used *Design patterns*, design models that solve a specific programming problem in a commonly accepted way [Kos05, page 102], are chosen to support the special requirements of mobile devices [Mik04, page 127]. One purpose of such design patterns is to help with the limited resources. Resources are controlled by the programmer instead of automatization such as garbage collection in Java programming language. Besides the memory, the battery of the device is also limited resource [Mik04, 127]. Symbian software frameworks supports modularization and code reuse. *Coding conventions* requires programmer to use special naming of variables, classes etc. to make the code understandable. Good example of such a naming convention is HBuf descriptor that tells to the user, that the memory for it is reserved from the Heap. Capital H in the beginning of the decriptor name signals that. Descriptors are introduced with other Symbian essentials in the next subsection.

3.1.1 Symbian essentials

One of the most fundamental of Symbian platform design decision is the optimization for efficient event handling. Symbian is designed to support event-based programming. Event-based design point of view derives from the Graphical User Interface (GUI) programming where the implemented program is dependant on the input of human [Tas00, page 96-97]. The same applies to socket programming as well. This differs from the Linux point of view (see Section 3.2). Programs are dependant of other systems, more precisely their input. Symbian has two main frameworks to support handling of events, Active Objects, and Client-Server architecture [Tas00, page 97].

Concurrent event handling can be done at many levels. Context change indicates the change of process and scheduling refers to changing of thread in execution. Symbian provides lighter way to handle concurrency with Active Objects framework inside a thread. It is event-based manner to manage execution and designed mobile devices in mind. Active Objects user space concept compared to the kernel-space context changes and thread scheduling. Implementation using Active Objects compared to threads, has been stated to end up to an order of smaller magnitude in time consumption [Mik04]. The Active Object framework is based on Active Scheduler that handles the event-based concurrency per thread. Active Scheduler is separate scheduler unit specific to this framework. Each of the Active Objects are installed to the Scheduler and when an event of an Active Object occurs, the RunL() -function is called. After this Active Object executes wanted functionality and stops or starts new asynchronous wait for another event. Several Active objects handles each of

them specific functionality of the implemented system in this manner. This approach does not allow new event for individual Active Object before the previous is ready. [Sym06] [Mik04]

Another fundamental design principle that is relevant to Symbian networking and event-based programming is called Client-Server design pattern [Kos05, page 136] [Sym06]. Symbian provides as one basic component of Inter-Process Communication (IPC) this framework. In this design pattern, multiple clients try to access resources that are controlled by a server. Motivation for such a design pattern is the encapsulation of resource management. Mutual exclusion and synchronization can be handled so that clients do not have to pay any attention on those [Mik04, page 56]. On Symbian, client and server run always in separated threads or processes. The channel between client and server is known as a session. An API defines the services for the client and by using different API, another set of services can be offered to the certain clients. In networking, this is natural way to implement software because of the fact that many services are controlled by a centralized management, such as servers.

Symbian has a careful point of view to the memory consumption because Symbian devices are intended to run long times without reboot. Thus, the avoiding of the memory leaks is crucial. It is also a problem with mobile devices that have limited amount of memory. There are two fundamental parts to handle memory efficiently, Cleanupstack, and descriptors. *Cleanupstack* is a Symbian specific data structure to prevent leaking of memory. Cleanupstack holds a pointer to an object that is dynamically reserved from the heap. In the case of using standard C++, memory leakings are possible if the pointers to the objects are lost. In case of cleanupstack, these memory blocks are released automatically if programmer forgets to do that. [Mik04, pages 145-147]

Descriptors are both fundamental to Symbian OS, and an excellent example of the difference in approach between non-Object Oriented and Object Oriented designs. *Descriptors* are a family of classes that are used in Symbian OS for string handling. All descriptor classes are inherited from the Base Class TDesC. Descriptors have some advantages compared to C-strings. They provide always inbuilt description of the amount of their memory allocation and the data structure, where they are allocated. Length is a data member of TDesC and name of a descriptor defines if it is allocated from the stack, heap, or Read-Only Memory (ROM) [Mik04, page 135]. Descriptors are used in preference to NULL-terminated C-strings. The same classes are used for general binary data. [Sym06] [Mik04]

3.1.2 Platform security

Symbian platform security provides background for a secure use of the system APIs. It also provides rules for secure software development. Some of the APIs are confidential and not seen by third parties.

The security model is designed for mobile devices. The idea is that a user has a device/devices. No one but the owner is using the same device by regular basis. In fact, there is no need to have several account or user rights. This is one cornerstone of the Symbian platform security design. The security model is process-oriented rather than user-oriented. In Symbian platform, rights and privileges are related to installed software components rather than to users unlike in Linux operating system. This is s a defense mechanism against malicious or badly implemented code. However, it is also a possibility to implement software for the platform by the authorized parties. The access of APIs or device resources by programs and approval of them are carefully considered. Another point of view is that mobile devices are used by variety of different users. There are no expectations of the level of knowledge of the users. A user may install software by accident or try to install malicious software. Symbian security model does process identification to install only authorized software. Rights, more precisely capabilities and process identification are described below. [Div05, page 4] [Hea06, page 23]

Installed software components have capabilities that describe the right to access sensitive resources of the device. Most important capabilities related to networking are NetworkServices and NetworkControl. NetworkServices allows the access to the services without any restriction on their physical location and NetworkControl gives the right to access or modify protocol controls. There are several other capabilities in the system, which does not relate to the networking [Sym06].

The identification of the application in execution is crucial part of the Symbian platform security. Capabilities reduce essentially the need for such an identification but there are circumstances, it is preferable to identify applications to be installed. Access Control List (ACL) of the device is constructed based on process identification. Every executable has unique Secure Identifier (SID) and it can contain a Vendor Identifier (VID). All of the applications have unique UID assigned by Symbian in addition to SID and VID. Those three identifiers are the basis of identification of the applications. Application must be signed and approved before allowing them to have individual identifiers. [Div05, page 9]

The APIs have classification system that divides them to the Published All, Published Partner, and Internal. Published All can be access any Symbian developer. Access to the Published Partner and Internal components is restricted. A license

is needed to access the published partner, and the internal components are internal for Symbian developers. [Sym06].

3.2 Networking in Symbian

Symbian networking differs from the Linux networking scheme. In Symbian, the socket API provides more extensive support for managing low-level functionality than in Linux. For example, interfaces are managed by the applications unlike in Linux where the user of the system needs to command the interfaces up and down before starting any applications.

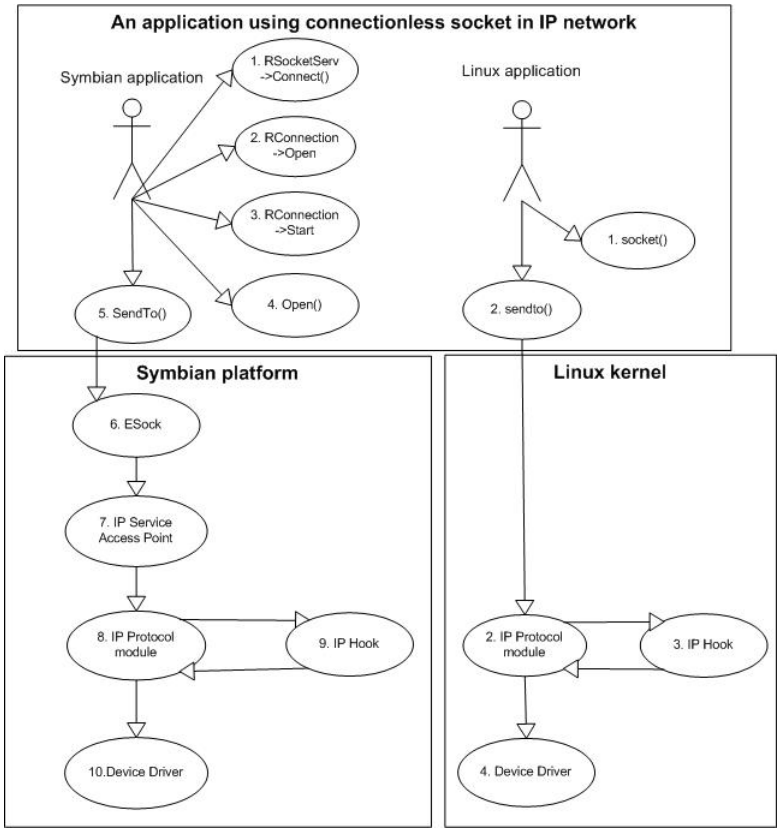

Figure 3.2: Connectionless socket send operation in Symbian and Linux.

Figure 3.1 illustrates the subsystems of Symbian version 9.2. Networking subsystem has infrastructure related to TCP/IP stack, Quality of Service (QoS) and IP over PPP protocol. It offers also secure sockets for Transaction Layer Security (TLS) and Secure Sockets Layer (SSL). Symbian TCP/IP stack offers sockets for IP, ICMP, TCP, and UDP protocols. It has also implementation of IPsec and DNS, which offers the host resolver to make DNS queries. [Sym06] [Cam07] [Sym05]

Comms Infrastucture is composed of communication database (CommsDat), ESock communication framework and network interface manager (NIFMAN). These are the controlling components of the networking infrastructure in the Symbian platform [Cam07, page 18]. CommsDat has the information of the Internet Access Points (IAPs) in a particular device. *Internet Access points* define the settings how a connection to the Internet is made [Cam07, 162] and it can be also understood as a single representative of a single network interface. NIFMAN, and ESock provide together the framework for managing the connections. NIFMAN is responsible for starting up interfaces including start-up of the default interface when there is no existing route to the next hop in the network. Interfaces can be controlled on the application layer via ESock that is the principal component of the Symbian networking. [Sym06] [Cam07] [Sym05]

The difference of Linux and Symbian socket API and the communication infrastucture are illustrated in the Figure 3.2. The difference from the application point of view is shown in the upper part of Figure 3.2. Symbian application uses three steps more to start communication with a connectionless socket. First, it has to connect to the socketServer, ESock. After this, it sets up the interface management. This is not necessarily needed if the interface management is not concerned by the application. The Symbian application opens and starts an interface (step 2 and step 3). Before sending anything both Linux (step 1) and Symbian (step 4) creates the socket instance.

The simplified presentation of the internal control in both Operating Systems is presented in the lower part of the Figure 3.2. Symbian communication infrastructure has more components in it. First, the socket server decides where to pass the control. In this particular case, it passes the control to the Service Access Point of the IP protocol, which is actually part of the IP protocol module. Both Linux and Symbian have hook mechanism, which is used to add certain type of protocol processing to the packets. After hooking both passes the control to the respective device driver in this simplified scenario.

3.2.1 ESock framework

ESock is a system wide way to access the different network resources. It provides the interface between a programmer and the communication modules. The three main services are:

- data protocols
- interface management
- name resolution

The data protocols are used for communications with a remote end. The implementation of HIP protocol module is based on the ESock protocol module framework (See Section 3.2.3). The interface management is the principal functional difference with the Linux environment. In Symbian, it is provided for the applications. Because of that, there is no information of the routes before a socket is opened. The name resolution i.e. DNS library is the third service provided by ESock. One advantage of having such a socket server, ESock, is the possibility to adapt legacy implementation to the new emerging technologies [Cam07, page 28].

ESock is implemented following the well-known Client-Server design pattern as introduced earlier in Section 3.1.1. Each protocol is running in a separate thread in Comms Process and ESock takes by the default care of forwarding the messages to right protocol. ESock server module, RSocketServ, provides services for the clients, RSocket and RConnection, connecting to it.

3.2.2 Socket API

Socket Application Programming Interface (API) is the client end point to a protocol. It is the Symbian variant of well-known Berkeley socket API. The socket API is protocol independent in both environments. Symbian Socket API has five main services:

- RSocketServ
- RSocket
- RConnection
- RHostResolver
- RNetDatabase

RSocketServ handle is needed to offer a channel according to the Client-Server architecture (Section 3.1) to communicate with socket server. Socket server coordinates the service requests from the applications to the responsible protocol module. Sockets must be always connected to a running socket server before they can be used. RSocketServ establishes an IPC communication channel for RSocket, RHostResolver, RNetDatabase and RConnection. RSocketServ introduces also methods for enumerating, finding, and stopping of protocols.

RSocket offers several services for creating connection, sending data, and receiving data. The usage, syntax, of such an RSocket is common for every protocol. On the other hand, the semantics of the services are based on the specific protocol and can be fine-tuned with ioctls options. Instead of blocking, sending and receiving operations are made asynchronously through RSocket. This allows application to stay responsive and react to received events meanwhile socket operation is waiting to be accomplished. Socket server monitors the amount of received data and when this condition is fulfilled, it fills buffers and signals the application. This is implemented by following the active object framework. Event waiting loop and handler dispatcher are provided and installed in the active scheduler. RSocket has four options how data is sent and received. Stream sockets are used for reliable byte-oriented channels. *Reliable* means that received bytes are in sending order without duplication or loss if received at all. Sequenced packet socket provides a reliable packet-based interface. It is guaranteed that packets blocks of data not bigger than the defined maximum length, are received in order if received at all. In this case, there is a maximum length for the packets, which must be respected on the sending side. Datagram socket is a packet-based interface for sending and receiving data. It can be reliable or unreliable depending on the underlying protocol. *Unreliable* means that the data can be received out-of-order or not at all. The fourth type, raw socket, provides an unreliable access to data transfer without restricting manipulation of the data.

Compared to Personal Computers mobile phones have much more variety on the IP Bearer technologies. *Bearer* means the technology, which is used to carry IP traffic. The multitude connection options available are handled with Symbian connection management. RConnection provides the API to take control over the connectivity at application level. Starting, stopping, and monitoring of IP Bearers are possible including the query mechanism that allows run-time choices to be made. RConnection can be used from multiple applications and multiple RConnections can be associated to a single Bearer technology. RConnection has also RSubConnection API that is used for Quality-Of-Service (QoS) management. A client may use RSubConnetions to set up a group of connection with different characteristics

for bandwidth, cost, or latency.

RHostResolver is a class to name resolution services, such as DNS queries. It can be used to obtain names by giving address, address by giving names and getting or setting local host name. RHostResolver is a client handle to the DNS server running separately.

RNetDatabase provides the possibility to add and query information from CommsDat. CommsDat is a preconfigured database that is used to map the IAPs to the Bearer technology. Connection preferences are set up at CommsDat and always a default connection is defined. This is an initialization operation. This default connection will be connected in a case where user does not define any preferences via RConnection.

3.2.3 Socket Server Protocols

ESock defines an interface for all socket type communications and defines a plug-in architecture for implementing particular protocols. Plug-in architecture is based on the fact that C++ allows abstract base classes and has virtual functions. The abstract base class defines the interface, but the implementation, *a plug-in*, can be added later on and changed over time. Protocol modules are by the socket server at runtime either explicitly or on demand as dynamically linked libraries. When a description file (.esk) is introduced for the ESock at the start up, it will load explicitly defined protocol modules. All such a protocol modules offering socket services must have implementation derived from CServProviderBase -class. The inherited class defines *Service Access Point* (SAP) through socket interface for the application willing to use it. The protocol module itself is derived from CProtocolBase -class. If a completely new protocol family is defined, CProtocolFamilyBase must be specialized for those purposes. Here *protocol family* means a collection of protocols closely related to each other. An ESock protocol module can also be a stack itself. [Sym06]

3.2.4 TCP/IP

TCP/IP stack is implemented in Symbian as a Socket Server protocol (Section 3.2.3) module more precisely it is a protocol family. It provides socket services for IP, TCP, ICMP, and UDP protocols. Other protocols are often bound to the TCP/IP stack and implemented as TCP/IP extensions.

TSockAddr is provided as a general socket address class for all the protocols in Symbian OS. It holds information of used port and the specialization adds more information along the child class. In case of TCP/IP family TInetAddr is such a child class that is derived from the TSockAddr. TInetAddr has an attribute for

IP address. Both versions of IP can be used with it by specifying the protocol family that will be used. If KAfInet is used the data structure will store 32 bit long address. In contrary, if KAfInet6 is used the data structure stores IPv6 address or IPv6 mapped IPv4 address. IPv6 mapped IPv4 address is an IPv6 address with special prefix and otherwise it saves the IPv4 address to the least significant bits. This can be also left undefined with KAFUnspec. [Sym06]

3.2.5 Security

Symbian Security subsystem is the base of Symbian security system. It implements cryptographical algorithms, hash functions, random number generator and supporting APIs. Cryptographical algorithms include encryption and decryption engines for both symmetric and asymmetric cryptography (Section 2.3). Integrity checking, signature verification and message digest functionality can be found from the Security subsystems as well (Section 2.3). Supporting APIs provides padding functions, arbitrary long integers, and password based encryption. These services are used by protocols and functions: Certificate Management, Software Installation, Secure Communication Protocols (SSL, TLS, IPSEC, and WTLS). WTLS is wireless version of Transport Layer Security. Some of these APIs are confidential.

3.3 Summary

As a conclusion, Symbian provides several software design solutions, which takes the nature of the mobile devices into account. Symbian design patterns help the programmer to design scalable and modular software, which try to take the scarce resources into account. Presently the amount of memory and etc. resources is increasing. However, Symbian provides challenges as well. Socket programming with Symbian differs from the well-known Berkeley sockets. The Symbian platform security provides challenges for user-space implementations. The heavy object-oriented architecture supporting variety of software design solutions may have some indications to the delay observed in the phone usage.

4. DESIGN

First, this chapter introduces the HIP protocol design. It describes the HIP PDU format and the state machine of the protocol. Then, the software architecture is on the focus. The structure of the relevant software components is introduced. Finally, the chapter summarizes the covered topics.

4.1 Protocol Design

The HIP design is simplier than the IKE design [Har98]. HIP does not have all the fine-grained details in the protocol, but instead of only being a key negotiation protocol, it supports host mobility and multihoming (see Section 2.4.4).

A HIP PDU consists of a HIP header and a combination of HIP parameters. The contents of PDUs vary depending on the usage of the optional HIP parameters. HIP has 19 different HIP parameters.

4.1.1 HIP PDU format

The HIP header is 40 bytes long as shown in Figure 4.1. It is logically an IPv6 extension header [Mos07, page 33]. The next header field derives from the IPv6 extension header. There is never next header, and thus 59 is the only supported value for the next header field to indicate that there is never next header. Packet type indicates the HIP PDU type. VER is the version field, which is at the moment one. RES is reserved for the future use. The checksum calculation uses a pseudo header. A UDP pseudo header is included in the case of IPv4. In the case of IPv6, the corresponding TCP/UDP pseudo-header is used. Controls are reserved for the future use except the last bit, which indicates that the HI of the sender in this PDU is unpublished. The parameter field is at the maximum 2008 bytes long. [Mos07, page 33-36]

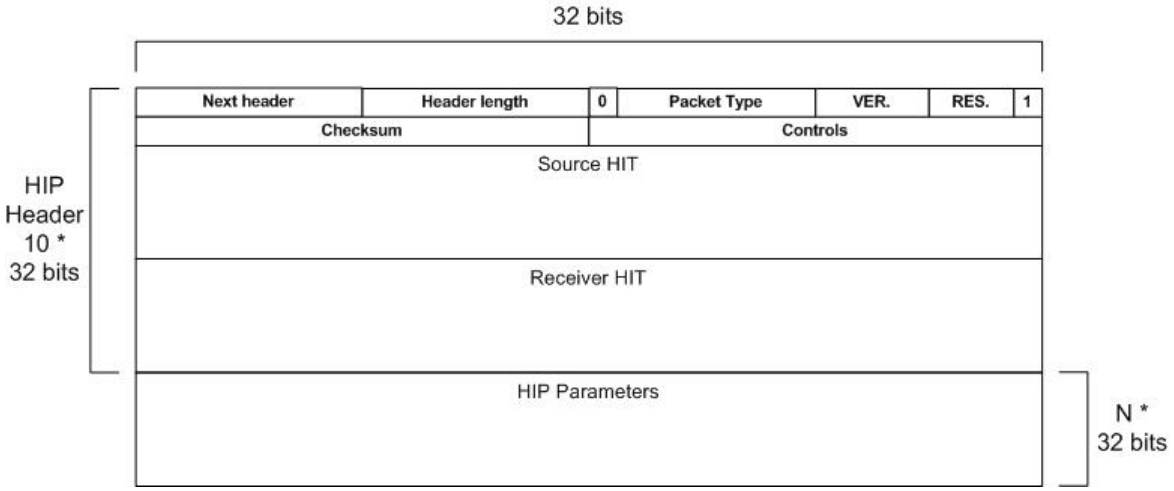

Figure 4.1: HIP PDU format.

4.1.2 HIP state machine

The HIP protocol has seven states and an error state called E-FAILED, which indicates that the HIP message exchange failed. There are 21 state transitions excluding the E-FAILED state. Initially the protocol is in the state UNASSOCIATED. After a successful Base Exchange (see Section 2.4.2), it establishes the connection and moves to the ESTABLISHED state. The closing happens if a timeout trigger launches for the connection. It sends the CLOSE-PDU and waits for a CLOSE_ACK-PDU. When a CLOSE_ACK-PDU is received or timeout takes place, the connection moves to UNASSOCIATED state (see [Mos07, page 30] for details).

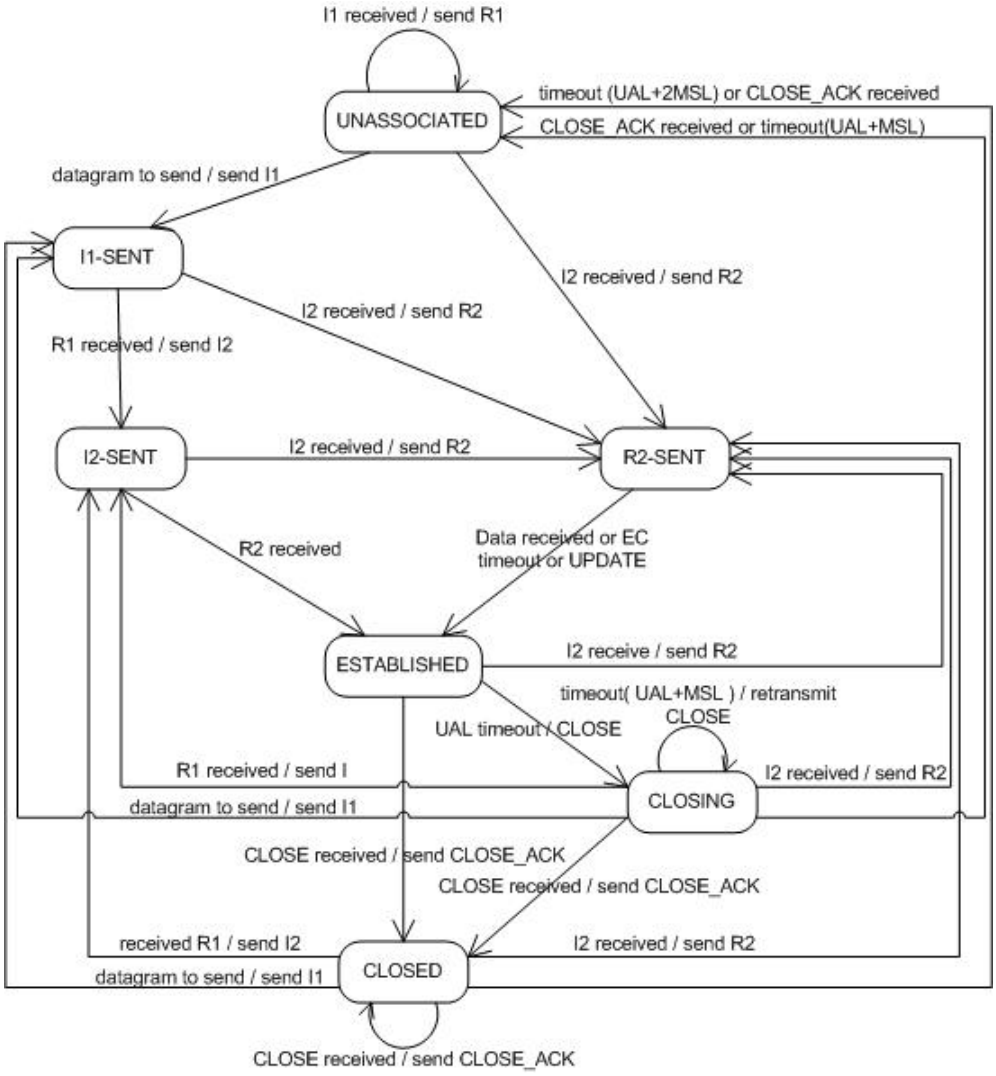

Figure 4.2: HIP state diagram

4.2 Software Design

The software design concentrates to the software architecture aspects of the implementation. The structure of the components, their interrelationships and principles and guidelines governing their design are important. The development of the software depends on those facts. Thus, the architecture of the software is studied closely.

The Symbian IKE has been used as a guideline for the software design. A manual keying application, implemented for the testing purposes of the IPsec at Nokia, provides the basis for the PFKEY and policy operations. The HIP software is implemented using the Symbian APIs, software design patterns provided by the Symbian subsystems.

4.2.1 Structural description of the system

The HIP software is part of the Symbian networking as shown in Figure 4.3. All grey modules are entirely implemented in the thesis. Partially grey modules are modified existing Symbian modules. The arrowhead describes the software module, which is utilized by the other software module.

A test application, the HIP protocol module, the IPsec BEET mode, and the HIP daemon are implemented in the thesis. The HIP daemon has also user interface for testing purposes. Applications use the socket API to access the data protocols, manage interface and for the domain name resolution. ESock provides the socket API and channels the socket actions to the correct protocol module. The HIP protocol module provides the HIP Service Access Point and adds the HIP protocol id to the IP packet header. The HIP protocol module passes the control to the TCP/IP stack, which passes the control to the IPsec module based on the local policy decisions. *Local policy* defines the filtering, which is applied to the outbound and inbound traffic. IPsec module has a crucial role in the SA and key management. When an application connects first time, there are no SAs for the IPsec that is used for the HIP data traffic. The HIP Daemon listens the PFKEY socket, which implements the Service Access Point for PFKEY messages. To obtain SAs IPsec sends PFKEY messages, which are processed by interested key management applications. The HIP daemon is such key management application. The HIP Daemon implements software for the HIP messaging. The Crypto module offers algorithms for the cryptographical functions, and the PKI service provides an API to access Public Key Infrastructure services.

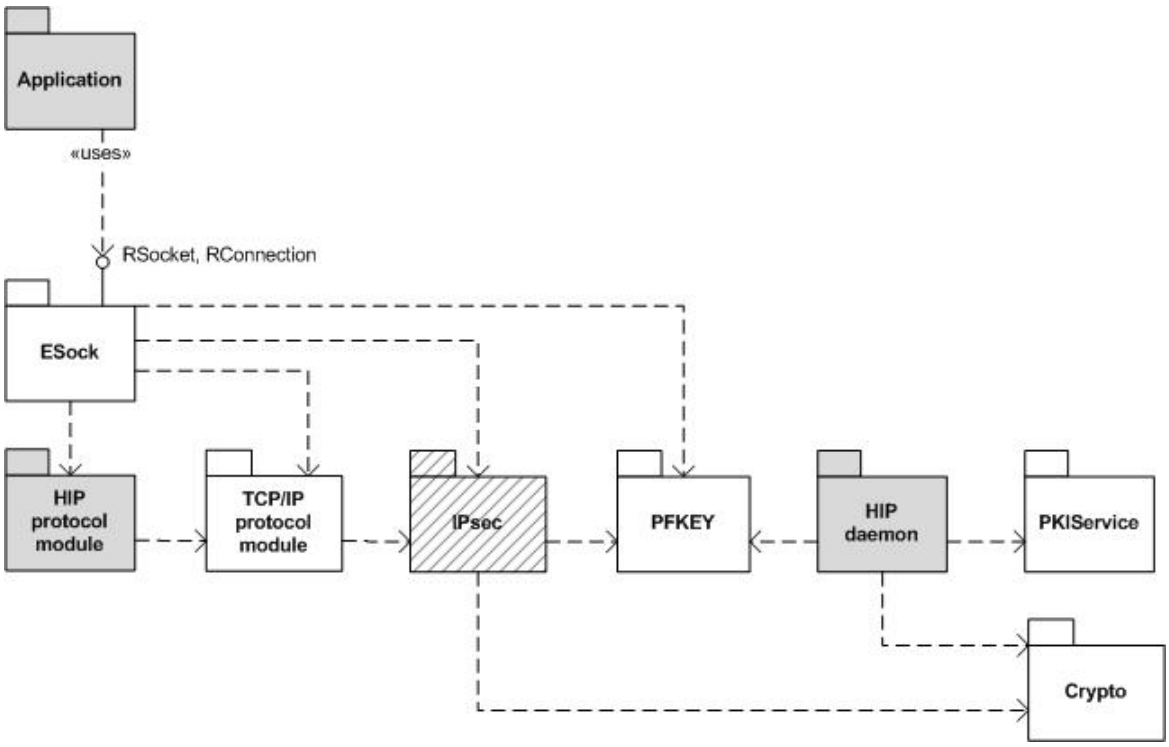

Figure 4.3: Package diagram of the core components related to HIP.

4.2.2 HIP daemon design

The HIP daemon is a standalone executable. It has integrated UI for the testing. The daemon itself works without the UI so the UI could be removed as a part of the future development. The daemon has four Active Objects, which handle the concurrency inside the daemon. The PFKEY-API, the socket sender, the socket receiver and the IPsec policy socket serve the engine asynchronously. A *software engine* is the core of a program that has the control of the other modules. Whenever a service completes, for example a message is received, it triggers the engine to pass the control for the responsible component. The engine controls the execution of the software based on the asynchronous events. The concurrency of the operations is achieved with the Active Object framework. The details are presented in the confidential appendix A.

4.2.3 HIP protocol module design

The HIP protocol module consists of classes that are inherited from the Symbian protocol module base classes (see Chapter 3.2.3). These base classes provide part of the functionality. In this manner, the fundamental behavior of the Socket Server Protocol modules is the same.

The HIP protocol module processes the socket API calls. The HIP protocol is used by utilizing a specific HIP protocol id (253) in the socket calls. The ESock installs the HIP protocol module when the Symbian starts up. The protocol module processes the calls and modifies the IP header so that it has HIP protocol ID in the protocol field.

The details are confidential and presented in appendix B.

4.3 Summary

The design of the HIP does not have all the fine-grained details that IKE has. Instead of that, HIP Base Exchange provides lighter option for the negotiations than IKE does. The software design has been made taking into account the quality aspects of the software architecture. The modularization and abstraction have been the main principles of the software design. Symbian provides natural environment for such design because it is object-oriented OS. However, the heavy utilization of the Symbian frameworks and the design patterns may cause overhead for efficiency to the implementation.

5. IMPLEMENTATION

This chapter introduces the functionality of the software. It describes the core parts implemented in the thesis: the HIP daemon and the IPsec changes. It also discusses of the experiences with the Symbian platform and the challenges of the environment for HIP implementation.

5.1 HIP daemon functionality

A start call from the user interface starts the engine of the HIP daemon. The HIP daemon engine sets up the PFKEY socket, and registers for the PFKEY messages. On the same occasion, the engine sets the socket receiver ready for the incoming HIP messages.

The Symbian IPsec and the HIP daemon are configured from the user interface. The local policy is defined in a separate policy file, which is loaded from the UI. The user must define the local policy in order to use the IPsec for the wanted data traffic. The UI triggers the engine to read the policy data from the IPsec policy file and to install the policy into the IPsec Security Policy Database. The HIP daemon listens and receives the PFKEY messages. The integration between HIP negotiations and PFKEY messaging is not fully implemented (see Section 6.3). The HIP Base Exchange negotiations do not implement HIP ESP extension [Jok06] for the SA negotiations. Instead of that, it provides possibility to install manually defined SAs to the IPsec. The UI provides the possibility to trigger the Base Exchange manually. By starting it manually, the sender sends I1 message, and stays listening incoming messages. The negotiations do not provide full processing of the HIP parameters. It implements the data structures for them, and the sending side for the HIP parameters. Full implementation would be future work (see Section 6.3). The confidential Appendix C shows the sequence diagram of a typical operation.

5.2 IPsec functionality

The IPsec works as a part of the TCP/IP stack. The IPsec is triggered based on the local policy decisions. *Local policy* refers to the filtering, which is applied to the outbound and inbound traffic. Application tries to connect and when the IPsec is

needed, TCP/IP stack passes the control to the IPsec module, which triggers a key management application to obtain proper SAs for the connection. After successful negotiations, the application sends message, which triggers the IPsec again. This time there are the SAs and the IPsec provides the wanted service for the payload: encryption or authentication. The trigger could be for example a socket API call connect(HIT), which connects to a remote Host Identity Tag. In this particular case, the actual IP header has different address than the socket call. The implemented IPsec BEET mode provides the functionality to build a new IP header for the routing.

The BEET implementation includes an engine for the processing inside the IPsec. In addition, to support it many changes had to be done. The IPsec processing of the matching traffic had to be changed. In the normal transport or tunnel mode, the traffic is matched with the IP addresses not with the identities. When the BEET mode is in use this is taken into account. The matching in the BEET mode has to be done with the identities instead of the IP addresses. The SA lookup had to be changed as well. The SA lookup is based on the the SPI and the IP addresses. Instead of that, with the BEET mode the lookup must be based on the SPI and the identities. Besides that, the PFKEY-API had to be extended to support the BEET implementation. The PFKEY message processing has several steps in the IPsec, and those are modified.

The details of the IPsec implementation are confidential. The sequence Diagram of the BEET functionality is presented in the confidential Appendix D.

5.3 PFKEY-API

The PFKEY is a socket protocol family used by trusted privileged key management applications to communicate with the key management internals of an operating system [McD98, page 3]. IPsec incorporates a Security Association Database (SADB), which is the place of storage for Security Associations. The PFKEY-API defines a well-known procedure to provide the information of SA changes between the IPsec SADB, and the key management. The PFKEY-API defines a set of messages, which are used for communication between the SA management and the key management as shown in table 5.1.

The key management application such as IKE or HIP register as listeners for the SA management with an SADB_REGISTER message as shown in Figure 5.1. If the SA management finds no SA for a new connection, it sends SADB_ACQUIRE message, which has the details of the needed SA. Such information as SA type, process id, associated addresses etc are passed to the key management application. Process

PFKEY-messages	
SADB_RESERVED	Reserved for the future use
SADB_GETSPI	Request/Response for obtaining SPI value
SADB_UPDATE	Request/Response for updating existing SA
SADB_ADD	Request/Response for adding SA
SADB_DELETE	Request/Response for deleting SA
SADB_GET	Request/Response for getting SA
SADB_ACQUIRE	Request from the kernel to acquire SA
SADB_REGISTER	Request/Response for key management application registration
SADB_EXPIRE	Request from the kernel to negotiate new SA
SADB_FLUSH	Request/Response to delete all the entries in the SADB
SADB_DUMP	Request/Response to dumb all the entries

Table 5.1: PFKEY-messages

id is UID in case of Symbian (see Section 3.1.2). The sequence diagram in Figure 5.1 illustrates the typical installation of unidirectional SA for a new connection.

The key management process gets the SPI value with the SADB_GETSPI -message and the SA management returns the SPI with the same message. After the key negotiations, the key management updates the SA to the mature state with SADB_UPDATE message.

The usage of PFKEY-API needs process identification. In case of Symbian, PFKEY messaging uses UID (see 3.1.2) instead of the process identifier.

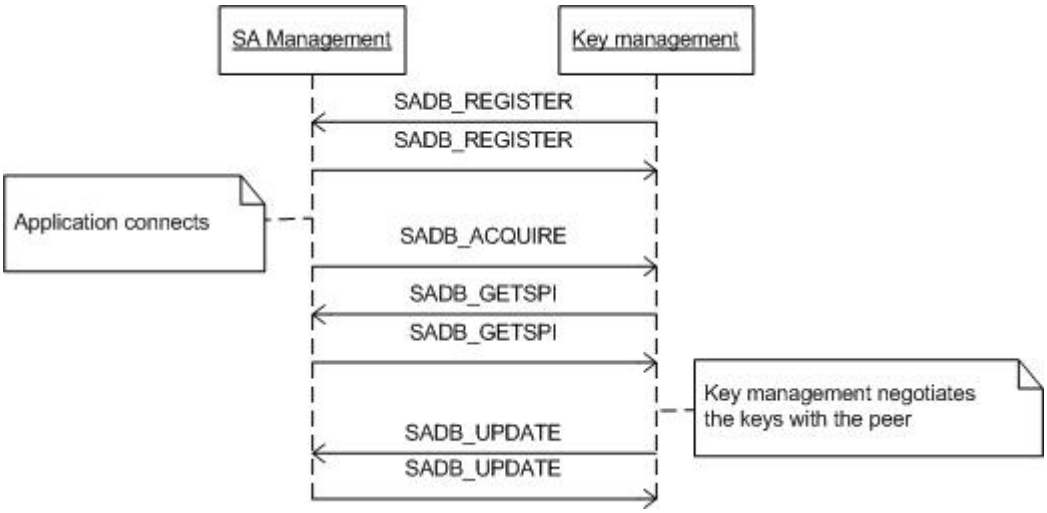

Figure 5.1: An example PFKEY message sequence for installing an SA.

5.3.1 Extended PFKEY-API

The well-known PFKEY-API version 2 [McD98] has to be extended the for IPsec BEET mode usage. These extensions were first introduced along the SHIM6-project [shi08]. The extension is also defined in the BEET draft [Nik07b]. The extension defines a new SA type, which stores the HITs or LSIs of both peers. The mode of the new SA is BEET. The mode must be defined in the PFKEY acquire messages. The implementation needed changes in the inner IPsec PFKEY message handling. The new SA type is called SA2.

5.3.2 Symbian implementation challenges

The Symbian APIs have challenges for the Symbian developers. According to the platform security, many of the APIs are classified to be non-public. Some of the internal APIs are limited and designed only for some specific purposes. For example crypto API has only a subset of the known cryptographical algorithms to offer.

Open C is a promising approach to implement Symbian software with the C-language using the POSIX libraries. When the project was started, Open C did not support IPv6, although the support has been recently added. Open C has the same support than the Symbian platform and is limited in that sense. One cannot extend the support without modifying Symbian. For example, POSIX standard socket interface does not provide support for undefined protocol id used with the HIP. Socket open call with the experimental HIP id (constant 253) is not supported without a new socket server module implementing the needed SAP in the Symbian stack.

5.4 Summary

The HIP daemon implements a modular software package, which is a good starting point for the full implementation. To support HIP, the IPsec had to be changed and the IPsec PFKEY processing had to be extended. The implementation is not straightforward because Symbian provides challenges for such implementations. Some of the APIs are limited in terms of the HIP usage. However, the platform provides extensive support for the HIP implementation.

6. DISCUSSION AND ANALYSIS

First, this chapter introduces the testing of the implemented software. The testing section concentrates on the functionality. Next, the quality aspects of the implementation are evaluated. The focus is on the evaluation of the software architecture. Then, this chapter describes the improvements and future development of the implementation. Finally, the chapter introduces related work.

6.1 Testing

Figure 6.1 describes the testing environment. There are two laptops connected to the Nokia Network with the LAN connection. Both have an S60 emulator. The S60 emulator emulates the Symbian environment, which includes the implemented IPsec and HIP modifications. The HIP daemon is started from the UI. WinPCap, a link-layer network access tool in the Windows environment, is used to establish connection between the S60 emulator and the real network. After both emulators have started the HIP daemon, daemons set up the virtual network interfaces with a static IP addresses or with the DHCP. A *virtual network interface* is a software abstraction, which does not necessarily associate with a physical network interface. Finally, both machines are ready to send or receive HIP packets.

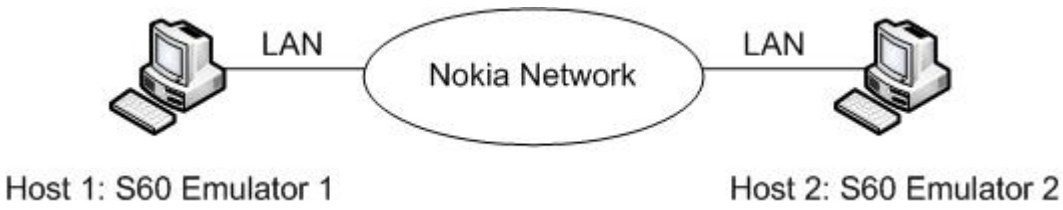

Figure 6.1: Testing environment.

Table 6.1: Test cases - group 1: IPsec BEET.

Case	Description	Outcome
Case 1.1	Install SA type 2	PFKEY operation succeeded. PASS
Case 1.2	Host sends to the peer's identity	IP packet sent. PASS
Case 1.3	Communicating hosts, LSIs IPv4 net	End-to-end messaging. Not tested.
Case 1.4	Communicating hosts, HITs IPv4 net	End-to-end messaging. Not tested.
Case 1.5	Communicating hosts, LSIs IPv6 net	End-to-end messaging. Not tested.
Case 1.6	Communicating hosts, HITs IPv6 net	End-to-end messaging. Not tested.

Table 6.2: Test cases - group 2: HIP daemon, HIP header.

Case	Description	Outcome
Case 2.1	Send correctly formed I1	BEX takes place. PASS
Case 2.2	Send I1, malformed Next Header field: value 0	Packet dropped. PASS
Case 2.3	Send I1, malformed Header Length: value 20	Packet dropped. PASS
Case 2.4	Send I1, malformed Packet Type field: value 20	Packet dropped. PASS
Case 2.4	Send I1, malformed Version field: value 0	Packet dropped. PASS
Case 2.6	Send I1, malformed Reserved field: value 1	Packet dropped. PASS
Case 2.7	Send I1, malformed Checksum field: value 0	Not implemented. FAIL
Case 2.8	Send I1, malformed Controls: value 2	Packet dropped. PASS
Case 2.9	Malformed HITs: value both HITs Zeros	Packet dropped. PASS

The testing of the implementation is done in the groups. The groups concentrate on certain part of the implementation. Some of the tested groups are presented in Table 6.1 and in Table 6.2. Both laptops have S60 emulators running and the HIP daemon is started on the both machines.

The first group tests the IPsec BEET mode. Before sending any traffic, the policy, and the PFKEY data is loaded to the IPsec policy management. Policies are managed separately from the SAs. After this, a HIP I1 packet is sent, and when the IPsec policy matches, the stack passes the control to the IPsec and then to the HIP daemon, which installs the manually defined SAs according to the identities for the inbound and the outbound traffic. The PFKEY processing modifications, the new Security Association type 2, and the IPsec changes concerning the BEET mode are tested this way. The test cases 3-6 test the BEET mode between two Symbian hosts, but were not executed due to time limitations of the project.

The second test group tests the basic HIP messaging with malformed HIP header values. The UI is used to initiate the Base Exchange in every case with different parameter values. The first case sends correctly formed HIP I1 packet and the Base Exchange proceeds. The following cases have each different malformatted field in the HIP header, and the packets are dropped in these cases.

Table 6.3: Quality attributes for the implementation.

Quality attribute	Description
Modifiability	How well the software supports large modification.
Maintainability	How well the software supports small modification.
Portability	How well it is possible to port the software.
Reusability	How well the software or parts of it can be reused.
Feasibility	How well the specifications can be followed in the implementation.
Interoperability	How well the software works in conjunction with other implementations.

6.2 Evaluation of the software architecture

The software architecture evaluation tries to identify the quality attributes of the software architecture [Kos05, page 222]. It does not directly try to evaluate how well the functionality corresponds the requirements. Instead of that, it tries to find the non-functional attributes that have an influence to the software quality.

The ATAM [Kaz00] method tries to evaluate the quality of the software architecture. The method was developed to provide a principled way to evaluate the fitness of the software architecture with the respect to the multiple competing quality attributes. The quality attributes, which are considered in this work, are introduced in Table 6.3 [Hai02, page 309].

The software architecture of the HIP Daemon is divided into the several pieces to improve the modular design and the abstraction. The module separation is based on the idea that the engine has the control of the software. In the Symbian, modules are often composed of several classes. Engine controls the inputs from the user, from the peer machine and from the IPsec and operates based on those. Negotiations are implemented in a separate class to make it possible to instantiate multiple of them. Every negotiation object manages its own state. Negotiation object constitutes of the connection objects, which is another principal class of the implementation. Every negotiation object may have multiple connections, which makes multiple connectivity scenarios possible. Connection module manages the sockets, which are implemented as separate modules: a sender and a receiver.

6.2.1 Modifiability

The negotiation class does not have association with the socket interface at all. This makes it possible to change the socket types. The UDP encapsulation for the HIP packets could be easily implemented in a separate socket modules. It is possible to implement another connection module, which has the UDP sockets for

the communications. This design choice gives negotiations possibility to choose the wanted method for the communication.

The connection class provides possibility to implement different connectivity scenarios. The connectivity is abstracted and thus negotiations do not know about the connectivity management. Because of that, routes and interfaces can be modified and managed inside the connection module.

One principle decision was that the protocol is not implemented as a plugin. A *plugin* is an application-specific extension module, which implements the APIs of the core module [Kos05, page 198]. The core module is the engine in this particular case. This principle decision restricts the usage of the engine as a core module for the different protocol implementations. The plugin architecture provides possibility to link the different implementation with the engine. The plugin architecture was given up due to the large amount of the coding it needs. However, it is not necessarily better solution to use plugins because of the overhead that plugins cause in terms of the code and the code efficiency. Because the plugins are loaded at the runtime, they cause overhead cycles for the CPU. The plugin architecture requires dynamic linking of the plugin to core module. The interactions between the core module and plugins may be heavier than implementing such a software without plugins.

6.2.2 Maintainability

The software does not fully implement the HIP protocol. However, adding of the new features is straightforward because the software architecture has well-defined responsibilities for each module. If the HIP parameters need changes to the implementation, these are made to the negotiations module. There is no need to modify other modules because they have different responsibilities.

Another aspect is the PFKEY messaging. Because the message handling is separated in the own module, the handling can be modified without modifications to the other modules. The encapsulation of the functionality guarantees that.

The clear naming conventions for the classes, functions, and variables improve the maintainability because the understanding of the software is achieved more easily.

6.2.3 Portability

Porting of the software between different Symbian phone families is possible. Because the UI layer differs in the Symbian families, it has to be removed from the Daemon, but the Daemon itself is a standalone executable. The UI in the software is implemented for the testing purposes.

6.2.4 Reusability

The code may be reused. The sender module and/or the receiver module can be easily moved to another software, if there is need to use Active Object -framework for the socket communications. Because the activities of the sender and the receiver are abstracted, another software component utilizing the components needs to only instantiate them, and the functionality is handled by the Active Object -framework. Thus only the Active Scheduler is needed to control the execution of the sender and the receiver. The connection class could be moved as a whole for the route and interface management, because it does not have association to the negotiations class, which owns it.

6.2.5 Feasibility

The software was a research prototype and the feasibility of the HIP was estimated during the implementation. The target was to produce information of the HIP in Symbian.

The HIP is not straightforward to implement. The IPsec internals have to be modified, which is time-consuming task. The modifications have to be made carefully, and the changes should be fully tested, because errors may have effect to the blocking of the unwanted traffic. The Host Identities should not leak on the wire in any form.

Besides the IPsec, Symbian provides a lot of work for the full HIP implementation. Some of the Symbian cryptographical APIs may be limited.

6.2.6 Interoperability

At the moment, interoperability with the other implementations is not tested. However, testing against own implementation seems to work with the implemented parts. The HIP parameters, which are not implemented, must be disabled.

6.2.7 Alternatives for the software architecture

It was chosen not to provide plugin implementation for the negotiations. That would have improved the modifiability, for example the HIP negotiation version would have been easy to choose at the runtime. However, because of the massive overhead the plugin causes in terms of code lines and code efficiency, it was abandoned.

The HIP was thought to be implemented as an ESock protocol module, but that seemed to be very disorganized design solution. The control would have gone

through the ESock protocol module to the IPsec and then back to the ESock protocol module from the IPsec. The ESock protocol module framework is designed mainly for the transport layer protocols. However, the implementation as a Esock protocol module may work by implementing the negotiations and the IPsec handling separately from the actual ESock protocol module classes.

6.3 Future work

The implementation is designed to be extended. Here are some points for the future development:

- The Daemon could be started by some other application instead of the UI. A related component could start the standalone executable at the start-up. The positive consequence is that the HIP negotiations would be triggered by a connecting application. However, this needs full integration of the PFKEY socket listener and the negotiations.

- The Daemon is limited in terms of HIP processing. Most of the BEX parameters are created but the processing of them is not fully implemented. The following HIP parameters should be implemented in the future: SIGNATURE, SIGNATURE2, SOLUTION, HMAC, HMAC2. The parameter data structures are implemented but the actual processing not. ([Mos07, pages 59-63])

- Checksum calculation should be included into the HIP header. This may be problematic because in the case of IPv6 the upper layer packet length must be known. In case of IPv4, the UDP pseudo header should be known and the problem may be the UDP length field.

- The protocol could be extended to support ESP [Jok06] and DNS [Nik07a] extensions. The ESP extension makes the exchange of SA information such as SPIs possible. The DNS extension defines an RR record for HIP. The DNS resolver would then perform a name to HIT lookup. In this approach, an application knows only a name of the other end.

6.4 Related work

At the moment, there are five different well-known HIP implementation. Ericsson Research Nomadiclab (FreeBSD), Helsinki University of Technology and HIIT (InfraHIP for Linux), Boeing Phantom Works (Linux), Andrew McGregor (Python user

level), Sun Labs Grenoble (Solaris). Besides these, HIIT has a Symbian implementation done in the user space.

6.5 Summary

The evaluation of the functional and non-functional requirements provides information for the future development. The functional requirements are evaluated by testing the software. The non-functional requirements are evaluated with the quality attributes. The architecture of the implemented protocol provides the starting point for implementing new subsets of HIP features. According to the observations of the implementation process, HIP may be easier to develop first apart from the IPsec, because the integration needs deep understanding of the Symbian internals and how the relevant parts have to be modified.

7. CONCLUSION

The focus of the thesis was the Host Identity Protocol and its suitability for the Symbian platform. The scope was to implement relevant parts of the HIP Base Exchange and to produce information for further development in the Symbian platform. The thesis covers the fundamental issues when a new protocol such as HIP is added to the platform. HIP is closely bound to the internals of the platform, and thus needs careful studying.

The author implemented a prototype, a daemon, which handles part of the protocol activities. It does not provide the whole solution, but instead of that, it implements a modular software package, which can be extended in the future. The respective software architecture of the implementation was designed during the project. The IPsec and the PFKEY processing were modified and a socket module was implemented to support HIP. Besides that, the relevant parts were tested with a simple testing setup, where two Symbian emulators communicated through Nokia network. A user interface component was implemented for the testing purposes. The non-functional requirements, such as software architecture were evaluated with the relevant quality attributes.

HIP provides a new namespace, which changes the semantics of the socket communications. Instead of the IPv6 addresses, applications bind to HITs, which are hashed public keys. Because of that, host multihoming and host mobility become easier. The new IPsec Bound-End-to-End Tunnel mode supports this by providing the mappings of identities and the IP addresses in the new defined SA type. The PFKEY defines the standard API for the SA management in the IPsec.

The Symbian platform provides extensive support for the modular software design and code reuse. However, the operating system provides challenges, because the internals of the OS are closely bound together and thus need careful studying before any modifications. In this case, several parts of the IPsec were modified. A new socket module had to be implemented as well because the platform did not support this recently defined protocol, HIP.

The observations made during the implementation process were that the complete HIP implementation is a very extensive area. HIP tries to solve several problems in the current Internet. It covers host mobility and multihoming, Internet security

and the problems with the current IP namespace. Besides the topics covered in this thesis, DNS, the HIP Rendezvous mechanism, NAT traversal, interface and route management need studying for a complete solution. Currently the implementation is limited in terms of such software scalability. However, it is a good starting point for a future implementation for the real mobile terminals.

BIBLIOGRAPHY

[Bos97] Bosch, J., Molin, P., Mattsson, M. & Bengtsson, P. Object-Oriented Frameworks Problems & Experiences. Research report ISSN 11031581, University of Karlskrona/Ronneby, Department of Computer Science and Business Administration, September 1997. 18 p.

[C02] Perkins C. IP Mobility Support for IPv4. RFC 3344, Internet Engineering Task Force, August 2002. 99 p.

[Cam07] Campbell, I. *Symbian OS Communications Programming.* John Wiley & Sons, Southern Gate, second edition, 2007. 444 p.

[Dee95] Deering, S. & Hinden, R. Internet Protocol, Version 6 (IPv6) Specification. RFC 1883, Internet Engineering Task Force, December 1995. 37 p.

[Dif76] Diffie, W. & Hellman, M.E. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[Div05] Dive-Reclus, C. Symbian OS v9 Security Architecture. Technical report, Symbian, February 2005.

[Dor99] Doraswamy, N. & Harkins, D. *IPSec: The New Security Standard for the Internet.* Prentice Hall, Upper Saddle River, 1999. 216 p.

[Hai02] Haikala, I. & Marijarvi, J. *Ohjelmistotuotanto.* Talentum, Helsinki, 2002. 430 p.

[Har98] Harkins, D. & Carrel, D. The Internet Key Exchange (IKE). RFC 2409, Internet Engineering Task Force, November 1998. 41 p.

[Hea06] Heath, G. *Symbian OS: Platform Security.* John Wiley & Sons, Southern Gate, 2006. 249 p.

[Hee06] T.M. Heer. LHIP: Lightweight Authentication for the Host Identity Protocol (HIP). Master's thesis, University of Tubingen, August 2006.

[Hen03] Henderson, T. Host Mobility for IP Networks: A Comparison. *IEEE Network Magazine*, 17(6):18–26, November 2003.

[Hen07a] Henderson, T. End-Host Mobility and Multihoming with the Host Identity Protocol. Internet draft "draft-ietf-hip-mm-05", Internet Engineering Task Force, March 2007. Expired draft. 48 p.

[Hen07b] Henderson, T., Nikander, P. & Komu, M. Using the Host Identity Protocol with Legacy Applications. Internet draft "draft-ietf-hip-applications-02", Internet Engineering Task Force, November 2007. Work in progress. 18 p.

[Jok06] Jokela, P., Moskowitz, R. & Nikander, P. Using ESP transport format with HIP. Internet draft "draft-ietf-hip-esp-06", Internet Engineering Task Force, June 2006. Expired draft. 37 p.

[Kau95] Kaufman, C., Perlman, R. & Speciner, M. *Network Security: PRIVATE Communication in a PUBLIC World.* Prentice Hall, Englewood Cliffs, 1995. 504 p.

[Kaz00] Kazman, R., Klein, M. & Clements, P. ATAM: Method for Architecture Evaluation. Technical report CMU/SEI-2000-TR-004 2367, Carnegie Mellon University, Software Engineering Institute, August 2000. 70 p.

[Ken98] Kent, S. & Atkinson, R. Security Architecture for the Internet Protocol. RFC 2401, Internet Engineering Task Force, November 1998. 66 p.

[Ken05a] Kent, S. IP Authentication Header. RFC 4302, Internet Engineering Task Force, December 2005. 34 p.

[Ken05b] Kent, S. & Seo, K. Security Architecture for the Internet Protocol. RFC 4301, Internet Engineering Task Force, December 2005. 101 p.

[Kos05] Koskimies, K. & Mikkonen, T. *Ohjelmistoarkkitehtuurit.* Talentum, Helsinki, 2005. 250 p.

[Kos06] Koskinen, J. Course Information Security material [WWW], August 2006. [referred 08.10.2007]. Available at: http://sec.cs.tut.fi/maso/.

[Kur07] Kurose, J.F. & Ross, K.W. *Computer Networking.* Pearson Education, Boston, fourth edition, 2007. 852 p.

[Lin06] Lindqvist, J. Establishing Host Identity Protocol Opportunistic Mode with TCP Option. Ietf internet draft "draft-lindqvist-hip-opportunistic-01", Internet Engineering Task Force, September 2006. Expired draft. 13 p.

[McD98] McDonald, D., Metz, C. & Phan, B. PF_KEY Key Management API, Version 2. RFC 2367, Internet Engineering Task Force, July 1998. 68 p.

[Mik04] Mikkonen, T. *Mobiiliohjelmointi.* Talentum, Helsinki, 2004. 248 p.

[Moc83] Mockapetris, P. V. Domain names: Concepts and facilities. RFC 0882, Internet Engineering Task Force, November 1983. 31 p.

[Mos06] Moskowitz, R. & Nikander, P. Host Identity Protocol (HIP) Architecture. RFC 4423, Internet Engineering Task Force, May 2006. 24 p.

[Mos07] Moskowitz, R., Nikander, P., Jokela, P. & Henderson, T. Host Identity Protocol Base Exchange. Internet draft "draft-ietf-hip-base-08, Internet Engineering Task Force, June 2007. Expired draft. 106 p.

[Nik07a] Nikander, P. & Laganier, J. Host Identity Protocol (HIP) Domain Name System (DNS) Extensions. Internet draft "draft-ietf-hip-dns-09", Internet Engineering Task Force, April 2007. Expired draft. 21 p.

[Nik07b] Nikander, P. & Melan, J. A Bound End-to-End Tunnel (BEET) mode for ESP. Internet draft "draft-nikander-esp-beet-mode-07", Internet Engineering Task Force, February 2007. Expired draft. 32 p.

[Per96] Perkins, C. IP Mobility Support. RFC 2002, Internet Engineering Task Force, October 1996. 79 p.

[Pos81a] Postel, J. Internet Protocol. RFC 0791, Internet Engineering Task Force, September 1981. 45 p.

[Pos81b] Postel, J. Transmission Control Protocol. RFC 0793, Internet Engineering Task Force, September 1981. 85 p.

[Rob67] Roberts, L.G. Multiple Computer Networks and Intercomputer Communication. Technical report, 1967. Proc. First Symp. on Operating Systems Prin., ACM.

[shi08] Site Multihoming by IPv6 Intermediation Working Group (WG), April 2008. Available at: http://tools.ietf.org/wg/shim6/.

[Sim95] Simpson, W. IP in IP Tunneling. RFC 1853, Internet Engineering Task Force, October 1995. 8 p.

[Sol04] Soliman, H. *Mobile IPv6: Mobility in a Wireless Internet.* Pearson Education, Boston, 2004. 338 p.

[Ste96] Stevens, W. R. *TCP/IP illustrated, Volume 1.* Addison-Wesley, Reading, Massachusetts, 1996. 576 p.

[Sym05] Symbian. Symbian OS Version 9.2 Technical specifications [WWW]. White paper, 2005. [referred 8.1.2008]. Available at: http://www.symbian.com.

[Sym06] Symbian. S60 3rd Edition SDK for Symbian OS v9.2, Supporting Feature Pack 1 [WWW]. SDK documentation, 2006. [referred 2.10.2007]. Available at: http://developer.uiq.com.

[Tan03] Tanenbaum, A.S. *Computer Networks*. Prentice-Hall International, Upper Saddle River, fourth edition, 2003. 891 p.

[Tas00] Tasker, M., Allin, J., Forrest, J., Heath, M., Richardson, T. & Shackman, M. *Professional Symbian Programming: Mobile Solutions on the EPOC Platform*. Wrox Press, Birmingham, 2000. 1031 p.

[Tho96] Thomson, S. & Narten T. IPv6 Stateless Address Autoconfiguration. RFC 1971, Internet Engineering Task Force, August 1996. 23 p.

[Zim80] Zimmermann, H. OS1 Reference Model-The IS0 Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1980.