HELSINKI UNIVERSITY OF TECHNOLOGY

Department of Computer Science

Bengt Sahlin

A Conduits+ and Java Implementation of Internet Protocol Security and Internet Protocol, version 6.

Master's thesis submitted for approval November 18, 1997 for the degree of Master of Science.

Supervisor Prof. Arto Karila

Instructor M.Sc. Pekka Nikander

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author and name of the thesis:

Sahlin, Bengt: A Conduits+ and Java Implementation of the Internet Protocol Security and Internet Protocol, version 6

Date: 18.11.1997

Number of pages: 66

Faculty:
Department of Computer Science

Professorship:
Computer Networks (Tik-110)

Supervisor: Prof. Arto Karila

Instructor: M.Sc. Pekka Nikander

As computer network technology advanced and the use of distributed systems became more common, new security problems arouse. Internet Protocol Security (IPSEC) is a proposal for providing security services in Internet. It offers authentication, integrity and confidentiality for IP datagrams. Internet Protocol, version 6 (IPv6) is the new version of the Internet Protocol. IPv6 offers expanded addressing capabilities, header format simplification, improved support for extensions and options, flow labelling capabilities and network security services. These security services are being offered through IPSEC. Conduits+ is a framework for building telecommunication protocol software. Its primary goal of supporting design of reusable object-oriented software is offered by building the framework using Design Patterns.

The goal of this project was to implement a prototype of IPv6 and IPSEC that is portable, reusable and modular. An object-oriented approach was taken and Java was chosen as the implementation language. The Conduits+ Framework was chosen as an experimental framework for the implementation. Main emphasis was put on the implementation architecture, which should be efficient and elegant. Performance was considered an issue only concerning the architecture, not the prototype. The IPv6 implementation would be minimal and key management was chosen to be manual.

The prototype was implemented in Sun Solaris 2.5. The specification work of IPSEC is in progress and the implementation is based on the Internet Drafts as they were in the beginning of the study.New drafts have been released and the prototype is therefore out of date.As part of the prototype, the Conduits+ framework was implemented in Java using the original Conduits+ ideas as the base and some new ideas by M. Sc. Pekka Nikander. An ethernet adapter was implemented in C code, and the implementation was tested in a real network between two hosts.

This thesis presents a Conduits+ architecture for the entire IPv6 and IPSEC. It seems to capture the dynamic and flexible nature of IPv6. Each IPv6 header is a module of its own. These modules all have the same structure. To support the arbitrary number of extension headers in IPv6, a loop is incorporated in the Conduits+ graph. The framework can be seen as memory efficient, as new objects, for example Sessions, are dynamically added. The thesis suggests breaking up the protocol functionality in relatively small parts, thereby mapping the solutions on the simple conduit types. The idea of breaking a protocol into elementary objects that capture the common characteristics of protocols, is a main strength of the Conduits+ framework.

Java offers portability. Any machine implementing the Java Virtual Machine can run Java byte code. Java offers libraries with useful methods for protocol programming. The garbage collection facilities free the programmer from memory management concerns and lessens the amount of bugs in the code. The use of native code in Java is not standardized yet and the scheduling policy is not precisely defined. Regarding telecommunications programming, adding an unsigned char type would be useful. The Java Developers Kit is not very mature and needs improvement. Performance can be significantly improved by using just-in-time compilers such as Kaffe.

Keywords: Internet security, TCP/IP, Internet, IPv6, IPSEC, Design Patterns, Conduits+, Java

Tekijä ja työn nimi:

Sahlin, Bengt: A Conduits+ and Java Implementation of the Internet Protocol Security and Internet Protocol, version 6

Päivämäärä: 18.11.1997 Sivumäärä: 66

Osasto:
Tietotekniikan osasto

Professuuri:
Tietokoneverkot(Tik-110)

Työn valvoja: Prof. Arto Karila

Työn ohjaaja: DI Pekka Nikander

Tietoliikennetekniikan edetessä ja hajautettujen järjestelmien yleistyessä uusia turvallisuusongelmia syntyi. Turvallinen IP, Internet Protocol Security (IPSEC), on ehdotus turvapalveluksi Internetiin. Se tarjoaa todennuksen, eheyden ja luottamuksellisuuden yhdistelmiä verkkoliikenteelle. Internet Protocol, version 6 (IPv6), on IP-protokollan uusi versio. IPv6 tarjoa laajennetut osoitemuodot, yksinkertaistetun otsakkeen, parempaa tukea laajennuksille ja valinnoille, tukea pakettien vuomerkinnälle ja tietoturvapalveluja verkkoliikenteelle. Nämä tietoturvapalvelut tarjotaan turvallisen IP:n kautta. Conduits+ on kehysympästö tietoliikenneprotokollasovelluksien rakentamiseen. Tukea uudelleenkäytettävyydelle oliopohjaisten ohjelmistojen rakentamisessa kehysympäristö tarjoaa suunnittelumalleihin (Design Patterns) pohjautuvan rakenteensa ansiosta.

Projektin tavoitteena oli rakentaa siirrettävä, uudelleenkäytettävä ja modulaarinen prototyypi turvallisesta IP:stä ja IPv6:sta. Lähestymistavaksi valittiin oliopohjaisuus ja toteutuskieleksi valittiin Java. Conduits+ kehysympäristö valittiin kokeelliseksi ympäistöksi prototyypin toteuttamiseen. Painopiste suunnattiin toteutuksen arkkitehtuuriin, jonka piti olla tehokas ja tyylikäs. Suorituskyky oli kriteeri arkkitehtuurille, ei itse prototyypille. IPv6-toteutuksen tuli olla minimaalinen ja avaintenhallinta manuaalinen.

Prototyyppi toteutettiin Sun Solaris 2.5-käyttöjärjestelmään. IPSEC:in standardointityö on kesken ja uudet ehdotelmat standardeiksi julkistettiin projektin aikana. Prototyyppi pohjautuu standardiehdotelmiin, jotka olivat voimassa projektin alkaessa. Osana projektia toteutettiin Conduits+ kehysympäristö Javalla pohjautuen alkuperäiseen Conduits+ kehykseen sekä DI Pekka Nikanderin uusiin ideoihin. Ethernetsovitin toteutettiin C-kielellä ja prototyyppi testattiin oikeassa verkossa kahden Sun-työseman välillä.

Tämä tutkielma esittelee Conduits+-arkkitehtuurin koko turvalliselle IP:lle ja IPv6:lle. Arkkitehtuuri vaikuttaa toteuttavan IPv6:n joustavan ja dynaamisen luonteen. Jokainen IPv6-otsake on oma modulinsa, ja jokaisella modulilla on samanlainen rakenne. Conduits+-graafissa käytetään silmukkaa tukemaan satunnaista määrää laajennusotsikoita, joita IPv6-otsakkeessa voi esiintyä. Kehys voidaan nähdä muistitehokkaana, koska uusia olioita lisätään dynaamisesti. Tutkielma esittää protokollan toiminnallisuuden pilkkomisen mahdollisimman pieniin osiin, jolloin ratkaisuihin voidaan käyttää Conduits+-kehysympäristön peruselementtejä. Conduits+-kehysympäristön yhtenä päävahvuutena voidaan pitää ideaa protokollan pilkkomisesta tiettyihin peruselementteihin, joilla mallinnetaan protokollien yleisiä piirteitä.

Java tarjoaa siirrettävyyttä. Missä tahansa koneessa, jossa on toteutettu Java virtuaalikone, voidaan ajaa Javan tavukoodia. Javan kirjastot tarjoavat hyödyllisiä metodeja protokollaohjelmointiin. Roskienkeruu vapauttaa ohjelmoijan muistinhallinnan ongelmista ja vähentää näin ohjelmien virheitä. Natiivikoodin käyttö ei ole standardoitu Javassa, eikä ajastamista ole tarkasti määritelty. Tietoliikenneohjelmoinnissa olisi hyötyä unsigned char-tyypistä. Java Developers Kit-ympäristö vaatii parannusta. Suorituskykyä voidaan parantaa käyttämällä Kaffen kaltaisia just-in-time-kääntäjiä.

Avainsanat: Tietoturva, TCP/IP, Internet, IPv6, IPSEC, Design Patterns, Conduits+, Java

Författare och arbetets namn:

Sahlin, Bengt: A Conduits+ and Java Implementation of the Internet Protocol Security and Internet Protocol, version 6

Datum: 18.11.1997 Antal sidor: 66

Avdelning: Data-avdelningen

Professur: Datornät (Tik-110)

Övervakare: Prof. Arto Karila

Handledare: DI Pekka Nikander

I takt med att datnätverksteknologin utvecklades och användningen av distribuerade system ökade, uppstod nya säkerhetsproblem. Internet Protocol Security (IPSEC) är ett förslag för att erbjuda säkerhetstjänster i Internet. IPSEC erbjuder autentikation, integritet och konfidentialitet för IP datagram. Internet Protocol, version 6 (IPv6) är den nya versionen av Internet Protocol (IP). Det erbjuder ökade addresseringmöjligheter, förenklat headerformat, ökat stöd för utvidgningar och optioner, flödeslabelmöjligheter och säkerhetstjänster för nätverk. De här säkerhetstjänsterna ebjuds genom IPSEC. Conduits+ är en ram (framework) för uppbyggandet av protokollprogramvara. Dess huvudsakliga mål - att stöda design av återanvändbar objektorienterad programvara - erbjuds genom att bygga upp ramen med hjälp av designmodeller benämnda DesignPatterns.

Målet med detta projekt var att förverkliga en portabel, återanvändbar och modulär prototyp av IPv6 och IPSEC. En objektorienterad infallsvinkel valdes och Java valdes till implementerinsspråk. Conduits+ ramen valdes som en experimentell ram för att bygga prototypen. Tyngdpunkt lades på implementeringen av arkitekturen, som skulle vara effektiv och elegant. Prestationsförmågan var ett kriterium enbart gällande arkitekturen, inte gällande prototypen. IPv6-implementationen skulle vara minimal och nyckelkontrolleringen manuell.

Prototypen implementerades i Sun Solaris 2.5. IPSEC-specifikationsarbetet pågår och nya protokoll specifikationsförslag har utgetts under projektets gång. Prototypen baserar de på Internet Drafts så som existerade vid början av projektet. Som en del av prototypen, implementerades Conduits+ ramen med hjälp av Java. Ramen baseras sig på de ursprungliga ideerna samt några nya ideer av DI Pekka Nikander. En ethernet adapter implementarades i C och prototypen testades i ett riktigt nätverk mellan två maskiner.

Detta arbete presenterar en Conduits+ arkitektur för hela IPv6 och IPSEC. Den dynamiska och flexibla IPv6-strukturen verkar passa bra in i arkitekturen. Varje IPv6 header är en självständig modul. Alla moduler har samma struktur. För att stödja den godtyckliga mängden för utvidgade headers, innehåller Conduits+ grafen en loop. Ramen kan anses vara effektiv i fråga om minnesavändning, och nya objekt, t.ex. sessioner, kan insättas dynamiskt. Erfarenheterna av projektet är, att det lönar sig att dela upp ett protokoll i relativt små delar. Härmed kan man basera implementationslösningarna på de elementära conduit-typerna. En betydande styrka i Conduits+ ramen är iden att tänka på kommunikationsprotokoll som bestående av vissa elementära delar, som beskriver de karakteristiska egenskaperna för protokoll.

Java erbjuder portabilitet. Java byte-kod kan köras på varje maskin som implementerar Java Virtualmaskinen (the Java Virtual Machine). Java erbjuder programbibliotek med användbara metoder för protokollprogrammering. Java erjbuder skräpsamling (garbage collection), som befriar programmeraren från problem gällande minneskontroll och minskar på mängden fel i programmen. Användningen av nativkod (native code) är inte standardiserad och skeduleringspolitiken är inte exakt definierad. Det skulle vara användbart att ha en unsigned char typ i protokollprogrammering. Java Developers Kit (JDK) kräver förbättring. Prestationsförmågan kan ökas märkbart genom att använda just-in-time kompilatorer, t. ex. Kaffe.

Keywords: Internet security, TCP/IP, Internet, IPv6, IPSEC, Design Patterns, Conduits+, Java

Acknowledgements

This master's thesis has been written as a research assistant in the Telecommunications Software and Multimedia (TCM) Laboratory of Helsinki University of Technology. During the work on the thesis, I have gained a growing interest in computer security matters. Through the courses thougt on computer security and computer networks in Helsinki University of Technology, and especially in the TCM Laboratory, I have got a profound basis on which to build my thesis.

I wish to thank my supervisor, Prof. Arto Karila for providing the opportunities to work with this topic, and for giving valuable comments in the completion phase. I wish to thank my instructor, M. Sc. Pekka Nikander for offering me a very interesting topic to write the thesis on, and for giving valuable comments through the project. I wish to thank my working colleague M. Sc.Timo Aalto for the valuable comments concerning IPSEC. My other colleague, Kaj Höglund, I wish to thank for the tight co-operation concerning Java and Conduits+. Also, I wish thank my fiancé Jonna for supporting me during the project and for her patience with my long hours working with an interesting and challening topic.

Table of Contents

List of Figures

Terms and Abbreviations

AH	Authentication Header
ATM	Asynchronous Transfer Mode
CBC	Cipher Block Chaining mode
Cryptographic Transformation	Cryptographic algorithm and set of rules surrounding the use of cryptographic algorithm in a particular context
Ciphertext	en encrypted message
Decryption	the proces of turning cyphertext into plaintext
DES	Data Encryption Standard. A symmetric key encryption and decryption algorithm
DLPI	Data Link Provider Interface
Encryption	the process of disguising a message in such a way as to hide its substance
ESP	Encapsulating Security Payload
Ethernet	A local area network technology
FTP	File Transfer Protocol
HMAC	Hash function based Message Authentication Code
Host-oriented keying	A keying approach in which all the users in the same host system share the authentication/encryption key
Host security policy	Expression of a security policy in a host system defined by an organization. Host security policy defines the security requirements to be fulfilled. It determines what security services should be used when communicating outside the host. An individual application or user may, in turn, modify their security requirements within the limits set by the security policy
ICMP	Internet Control Message Protocol

IDEA	International Data Encryption Standard
IETF	Internet Engineering Task Force. A group of people closely connected to the IAB who work on the design and engineering of TCP/IP and the global Internet
IGMP	Internet Group Management Protocol
Hash Function	A function that maps a variable-length data block or message into a fixed-length value called a hash code
Host ID	Host identifier
IP	Internet Protocol
IP association	5-tuple that consists of 1) local host IP address, 2) foreign host IP address, 3) local port number, 4) foreign port number, 5) protocol identifier
IP security policy	Host security policy applied to the IP layer security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IPSEC	Internet Protocol Security
ISAKMP	Internet Security Association and Key Management Protocol
IV	Initialization Vector. A random block of data that is used to begin the encryption of multiple blocks of plaintext when block-chaining encryption technique (e.g. CBC) is used
kb/sec	Kilo bits/second
kB/sec	Kilo bytes/second
Keyed MD5	MD5 algorithm used with a secret key
Keyed SHA	SHA algorithm used with a secret key
MAC	Message Authentication Code. An authenticator that is a cryptographic function of both the data to be authenticated and the secret key.

	Also referred as a cryptographic checksum
MD5	Message Digest algorithm 5. A one-way hash algorithm
MTU	Maximum Transfer Unit. The largest amount of data that can be transferred across a given physical network. The MTU is determined by the network hardware
Native Code	code written in a “native” language, usually C or C++, and can be invoked from Java
Network ID	Network Identifier
Oakley	The Oakley Key Determination Protocol
PGP	Pretty Good Privacy, an application providing confidentiality and authentication service that can be used for electronic mail and file storage applications
Public-key Encryption	A form of cryptosystem in which encryption and decryption are performed using two different keys, one of which is referred to as the public key and one of which is referred to as the private key
RFC	Request For Comments. The name of a series of notes that contain surveys, measurements, ideas, techniques, and observations, as well as proposed and accepted TCP/IP protocol standards
rlogin	The remote login protocol developed for UNIX by Berkeley
RIP	Routing Information Protocol
RSA algorithm	A public-key encryption algorithm based on exponentiation in modular arithmetic
Security Association	A Security Association (SA) is a relationship between two or more entities that describe how the entities will utilize security services to communicate securely. This relationship is represented by a the set of security information relating to a given network connection or a set of connections

Security Mechanism	A mechanism that is designed to detect, prevent, or recover from a security attack
SHA	Standard Hash Algorithm. A one-way hash function
SNMP	Simple Network Management Protocol. A standard protocol used to monitor hosts, routers, and the networks in which they attach to
socket	The abstraction provided by UNIX operating system that allows an application program to access the TCP/IP protocols
Solaris 2.5	Version 2.5 of an operating system developed by Sun Microsystems
SPI	Security Parameter Index. An unstructured opaque index which is used in conjunction with the IP destination address to identify a particular security association
SKIP	Simple Key Management for Internet Protocols
SSH	Secure Shell
STREAMS	A set of tools for developing system communication services
Symmetric Encryption	A form of cryptosystem in which encryption and decryption are performed using the same key
TCP	Transmission Control Protocol
TCP/IP	Acronym for the TCP/IP protocol suite
Telnet	The TCP/IP standard protocol for remote terminal service
UDP	User Datagram Protocol
UID	User Identifier in the operating system context
User-oriented keying	A keying approach in which users in a same host system have each separate keys in use

1 Introduction

1.1 Project Background

In the year 1995, a security project was started by M. Sc. Pekka Nikander in the Telecommunication and Multimedia laboratory at Helsinki University of Technology. The project deals with strong identification and access control in a distributed heterogeneous environment. The theoretical goal of the project is to implement strong authentication and access control in a distributed environment so that the following requirements are fulfilled:

- The security of the services does not depend on the security of the network.
- The security functions can be considered secure, if the cryptographic algorithms used can be considered secure.
- The failure of security in one part of the environment does not weaken the security in the rest of the environment.

As part of the project, some implementations are done. M. Sc. Timo Aalto has made a prototype of Internet Protocol Security (IPSEC) in the Sun Solaris 2.5 environment as a part of his master's thesis, which was completed in August 1996 [AAL96]. Based on this prototype, a beta version is currently being worked on by Santeri Paavolainen. A prototype of the key management protocol Internet Security Association and Key Management Protocol (ISAKMP) has been implemented as a part of the master's thesis of M. Sc. Kaj Höglund.

The practical work of the project also includes building prototypes of IPSEC in other environments. IPSEC is being implemented to Apple Macintosh by a group of students as a project in the Individual Project's course. This thesis describes an implementation of IPSEC using an object-oriented framework called Conduits+ and using the

Java language as the implementation language. A minimal prototype of IPv6 is implemented to be able to test the IPSEC implementation in a real network.

1.2 Goals

The main goals of this study was to build a portable, reusable and modular IPSEC prototype. To achieve these goals, an object-oriented approach was taken and the Java language was chosen as the implementation language. As part of the study, the suitability and maturity of Java would then be evaluated.

Main emphasis was put on the design of the implementation architecture, which should be efficient and elegant. Common, known object-oriented problem solving methods were intended to be used. This motivated the use of Design Patterns [GHJV95]. The Conduits+ Framework [EHJ96], which incorporates many Design Patterns, was chosen as an experimental framework for building the architecture. IPSEC was incorporated to the IPv6 protocol, so a minimal prototype of IPv6 was also implemented. Performance was stated as a goal only for the architecture, not for the implementation.

Based on the prototype, evaluations about the Conduits+ framework is presented in this study. Considerations about reusability is discussed.The suitability of the Java language for implementing telecommunication software is evaluated.

1.3 Limitations

Efficiency was considered an issue only concerning the architecture, not the actual prototype. The IPv6 implementation would be minimal and the key management was chosen to be manual.

1.4 Study Outline

The rest of this study is organized as follows . Chapter two is an introduction to computer security and security in Internet. Chapter three presents an introduction of Internet Protocol Security and IPv6. Chapter four presents object-oriented programming tec hniques . The J a va Language , Design P atterns and the Conduits+ framework are presented shortly . Chapter five presents the Conduits+ arc hitecture of IPv6 and IPSEC . The design c hoices of the arc hitecture are motivated. Chapter six presents results and experiences gained in the project.

2 Introduction to Network Security

Information security is an important matter in organizations. Before the widespread use of data processing equipment, information that was seen to be valuable to an organization was primarily secured by physical and administrative means. [STA95]

With the introduction of the computer, the need for automated tools for protecting files and other information stored on the computer became evident. This is especially the case for a shared system and fore systems that can be accessed over a data network. The generic name for the collection of tools designed to protect data and to thwart crackers is computer security. [STA95]

2.1 Basic Concepts of Security

As computer hardware became cheaper and the components shrank in size, and as network technology advanced, the usability and availability of computer services exceeded. However, the technologies lead to new **threats** against computer systems and new security matters had to be thought of.

2.1.1 Security threats

"A **threat** to a computer system will be defined as any potential occurrence, malicious or otherwise, that can have an undesirable effect on the assets and resources associated with a computer system". [AMO94]

The threats to computer systems can be categorized as follows:

"The **disclosure** threat involves the dissemination of information to an individual for whom that information should not be seen." [AMO94]. Much information flows through the computer networks. Anyone who has access to a node of a network, can listen to the traffic flowing through the node. If the data transmitted is not protected, some individual can get access to information that is not meant for him.

"The **integrity** threat involves any unauthorized change to information stored on a computer system or in transit between computer systems." [AMO94] It is of course important that no one can modify critical information transmitted in the network.

"The **denial of service** threat arises whenever access to some critical computer system resource is intentionally blocked as a result of a malicious action taken by another user." [AMO94]

2.1.2 Vulnerabilities

"A **vulnerability** of a computer system is some unfortunate characteristic that makes it possible for a threat to potentially occur." [AMO94] To mitigate threats, it is important to identify and remove vulnerabilities of the computer system. A weak or nonexistent password on some user account might for example be a vulnerability on a system.

2.1.3 Security attacks

"An **attack** on a computer system is some action taken by a malicious intruder that involves the exploitation of certain vulnerabilities on the part of an attacker." [AMO94] A successful attack leads to the occurrence of a threat to the computer system.

Attacks can be divided into passive attacks and active attacks. Active attacks involve modification of a data stream or creation of a false stream, while passive attacks do not involve modification of data. [STA95]

Masquerade and replay are examples of active attacks. A masquerade takes place when one entity pretends to be a different entity. Replay involves the passive capture of a data unit and subsequent retransmission of the data to produce an unauthorized effect. [STA95]

Eavesdropping and traffic analysis are examples of passive attacks. The goal of the eavesdropper is to obtain the contents of some transmitted data. The purpose of traffic analysis is to deduce useful information for the enemy by analyzing network traffic flow. [STA95]

2.1.4 Cryptography

To protect the users of computer systems from different kinds of threats, cryptography is used. The art and science of keeping messages secure is called cryptography. The art and science of breaking ciphertext is called cryptanalysis. The branch of mathematics encompassing cryptography and cryptanalysis is called cryptology. [SCH96]

"A message is plaintext. The process of disguising a message in such a way as to hide its substance is encryption. An encrypted message is ciphertext. The process of turning ciphertext back into plaintext is decryption." [SCH96]

There are two kinds of cryptographic schemes, symmetric and asymmetric schemes. Symmetric cryptography means that the parties use the same key; the sender encrypts with the key and the receiver decrypts with the key. Asymmtetric cryptography means that different keys are used for encryption and decryption. Any party has a secret key and a public key. The secret key can not be derived from the public key. The message is encrypted with the public key and only decrypted with the secret key. [SCH96]

"A trade-off exists between security and usability. A conflict generally occurs when the goal of information and resource sharing is combined with the goal of strict security controls between users. A worthy goal in the design and development of security solutions is to reduce the degree to which increases in security affect the usability of a system." [AMO94]

2.2 Security Protocols

[SCH96] says: "A protocol is a series of steps, involving two or more parties, designed to accomplish a task. A series of steps means that the protocol has a sequence, from start to finish. Involving two or more parties means that a protocol involves communication of some kind. This means that one person alone does not make a protocol. A protocol can be seen as an algorithm involving two or more parties communicating with each other".

There are a few important characteristics for a protocol: [SCH96]

- Everyone involved in the protocol must know the protocol in advance.
- Everyone involved in the protocol must agree to follow it.
- The protocol must be unambiguous.
- The protocol must be complete

Cryptographic protocols are communications protocols that aim to provide security services to the communicating parties or other protocols. They use cryptographic transformations, e.g. encryption, to the messages in order to ensure the confidentiality, integrity or some other aspect of the communication. A non-cryptographic protocol attempts to conduct communication in the presence of random errors, while a cryptographic protocol must ensure the prevalence security properties even in the face of a malicious, well designed attack. [NPSH97]

2.3 Internetworking Security Services

The introduction of distributed systems and the use of networks is a major change that affects security. Network security measures are needed to protect data during their transmission. In fact, virtually all organizations interconnect their data processing equipment with a collection of interconnected networks. Therefore, the term Internetworking security is also used when talking about network security. [STA95]

Security services enchances the security of the data processing systems and the information transfers. "The service counter attacks and makes use of one or more security mechanisms to provide the service" [STA95]. The following security services for internetworking are defined:

Authentication is "a security property that ensures that the origin of received data is, in fact, the claimed sender". [I-D96a] An example of authentication is using passwords, although it is a weak authentication and not considered very secure.

Integrity is "a security property that ensures data is transmitted from source to destination without undetected alteration" [I-D96a]

Confidentiality is "a security property that ensures that communicated data is disclosed to intended recipients, but is not disclosed to other, unauthorized parties" [I-D96a]

Non-repudiation is "a security property that ensures that a participant in a communication can not later deny having participated in the communication. This property may apply to either the sender or recipient, or both". [I-D96a]

In the context of network security, **access control** is the ability to limit and control the access to a host system and application via communicating links.

"The **Availability** A(t) of a system is defined as the probability that a system is operational at a time t or, in other words, the percentage of operational lifetime a system performs its functions." [TRI] [SS92] In other words, availability means that the intended services of a computer system can be offered in reasonable time. A variety of attacks can result in the loss of or reduction in availability. Some of these attacks can be defended by automated countermeasures such as encryption, others require some sort of physical action to prevent or recover from loss of availability. [STA95]

3 IPv6 and Internet Protocol Security

"The TCP/IP protocol suite allows computers of all sizes, from many different computer vendors, running totally different operating systems, to communicate with each other. If forms the basis for what is called the worldwide Internet, or the Internet, a wide area network of more than one million computer that literally spans the globe." [STE94]

This chapter first gives a short introduction to the TCP/IP protocol suite. Then, IPv6, Internet Protocol Security (IPSEC) and Key Management is discussed in more detail.

3.1 The TCP/IP Protocol Suite

Application	Telnet, FTP, e-mail, etc.
Transport	TCP, UDP
Network	IP, ICMP, IGMP
Link	device driver and interface card

FIGURE 1. The four layer of the TCP/IP protocol suite.[STE94]

The TCP/IP protocol suite is divided into four layers. These are the application layer, the transport layer, the network layer and the link layer. On each layer, there is a number of different protocols. [STE94]

The link layer normally includes the device driver in the operating system and the corresponding network interface card in the computer. They handle all the hardware details of physically interfacing with the cable. The device driver could for example be an Ethernet driver. [STE94]

The network layer handles the movement of packets around the network. IP (Internet Protocol), ICMP (Internet Control Message Protocol, and IGMP (Internet Group Management Protocol) provide the network layer in the protocol suite. [STE94]

The Transport layer provides a flow of data between two hosts, for the application layer above. The protocols belonging to this layer are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). TCP provides a connection-oriented, reliable byte stream service. UDP is a simple, datagram-oriented, connectionless protocol. It provides no reliability, there is no guarantee that the packets ever reach their destination. [STE94]

The application layer handles the details of the particular application. The most common applications are: Telnet for remote login, FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol) and SNMP (Simple Network Management Protocol). [STE94]

3.2 IPv6

"The function or purpose of Internet Protocol is to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one internet module to another until the destination is reached" [RFC791].

"The Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks. The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses" [RFC791].

IP provides an unreliable, connectionless datagram delivery service. Unreliable means that there are no guarantees that an IP datagram successfully gets to its destination. IP provides a best effort service. [STA95]

IP version 6 is a new version of the Internet Protocol. It is planned to be the successor of IPv4, which is the protocol used in the Internet today. When planning IPv6, the approach was taken that IPv4 is a good design, and therefore IPv6 should keep most of the characteristics [HUI96].The primary changes from IPv4 to IPv6 are the following: [RFC1883]

- **Expanded Addressing Capabilities**. The address size is increased from 32 bits in IPv4 to 128 bits in IPv6 to support more levels of addressing hierarchy and a much greater number of addressable nodes. Also, a new type of address called an anycast address is defined. It is used to send a packet to any one of a group of nodes.
- **Header Format Simplification**. Some of the IPv4 header fields have been dropped or made optional. This simplification aims to reduce the processing cost of common-case packets and to limit the bandwidth cost of the IPv6 header.
- **Improved support for extensions and options.** Extensions and options are incorporated in IPv6 packets as new headers after the IPv6 header. This leads to more efficient forwarding, less stringent limits on the length of options and greater flexibility for introducing new options in the future.
- **Flow labelling capability**. This new capability aims to enable the labeling of packets belonging to particular traffic "flows" for which the sender requires special handling. This could be for example "real-time" service. Flow labelling is still experimental and subject to change as the requirements for flow support become clearer.

- **Authentication and Privacy capabilities**: The security services in IPv6 are being offered through Internet Protocol Security (IPSEC). Thus, IPSEC has to be implemented as part of IPv6.

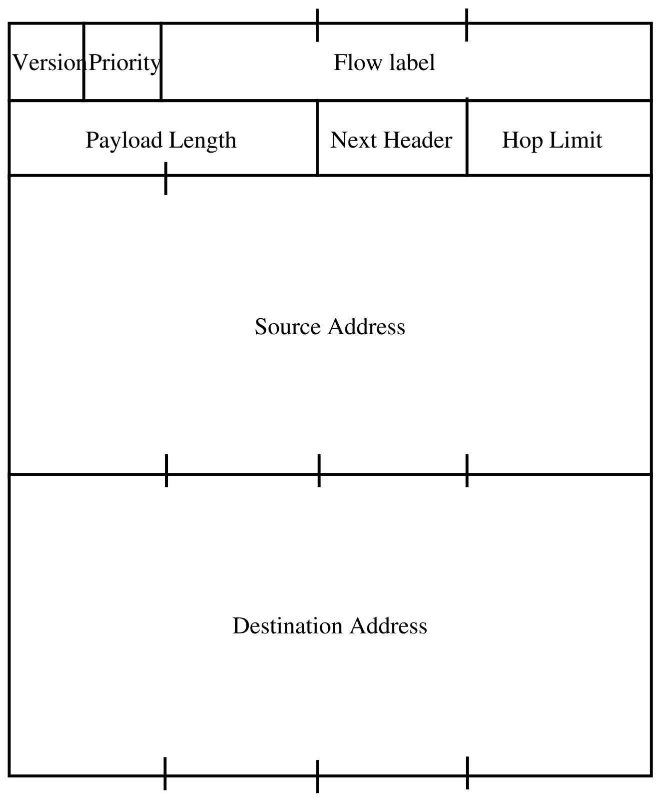

FIGURE 2. The IPv6 Header Syntax [HUI96]

Figure 2 shows the Header format of IPv6. The version is the 4-bit Internet Protocol version number, in this case six. The following four bits are the Priority value. This value identifies the desired delivery priority on its packets. The 24 bit flow label is an identifier for packets requiring special handling as explained earlier. The 16 bit payload length tells the length of the data that is carried in the IPv6 packet. Naturally, no header length needs to be known, as the IPv6 Header is constant in length. The Next-Header field is a 8-bitfield that tells what kind of header comes next in case there are extension headers. Instead of Time-to-live, the IPv6 packet has a Hop Limit field. This field is decremented by 1 by each node that forwards the packet. The last parts of the packet are the Source address and Destination address.

"In IPv6, it is possible to insert an arbitrary number of extension headers between the Internet Header and the Payload. Each header is identified by a header type and carries the header type of the following header in the chain, or that of the payload in the case of the last extension" [HUI96].

The current IPv6 specification defines six extension headers:

- Hop-by-Hop options header
- Routing Header
- Fragment Header
- Authentication Header
- Encrypted Security Payload
- Destination Options Header

The philosophy of IPv6 is, that all the ordinary packets should be routed through the network with as less processing as possible. Therefore, almost all special handling of packets, for example fragmentation, is processed through the extension headers.

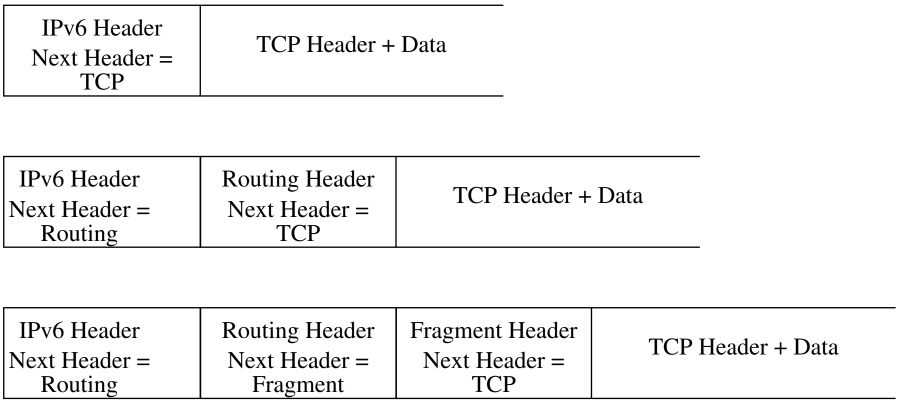

FIGURE 3. Extension Headers Examples [HUI96]

Figure 3 shows IPv6 header examples. The first one is an ordinary packet without any extension headers. The second example shows a packet with an Routing Header. The extension Header is inserted between the IPv6 Header and the upper layer protocol data. The third example shows an additional Fragment Header inserted in the packet.

3.3 Internet Protocol Security

The development of security services in Internet can be approached in many ways. Internet Security Protocol (IPSEC) is a proposal, where the services are placed in the network layer. Network layer security has three advantages. First, if security is available at this level, it becomes a standard service for all applications to use. Second, the complexity of cryptographic systems gives a good reason for applications to use an externally provided component, so that every application does not need to implement the services. Third, some attacks can be thwarted only at the Internet Layer. [HUI96]

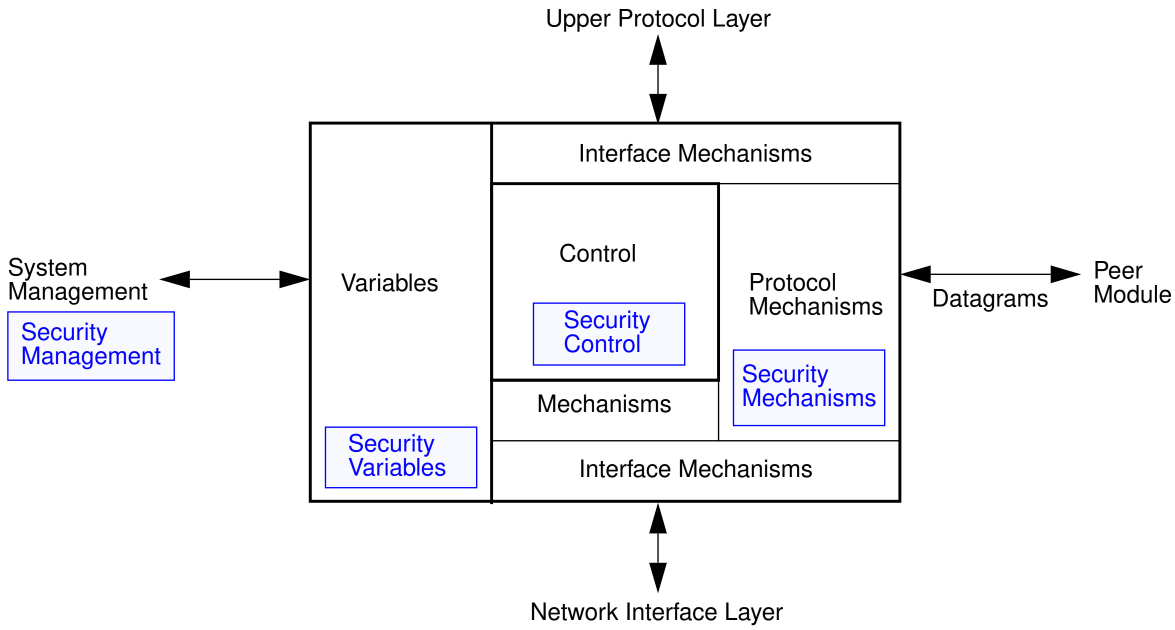

FIGURE 4. Conceptual model for the IP security architecture. [KAR91]

Figure 4 shows a conceptual model for the IP Security architecture. IPSEC is still an Internet Draft, and the specification work is still on progress. The general conceptual model is presented in [KAR91] and applied to IPSEC architecture in [AAL96]. Using the concepts of this model, the IPSEC architecture consists of security variables, mechanisms, control and management.

3.3.1 Security Variables

"The security variables include the information controlling the selection and use of various security mechanisms and contain information about how to apply security mechanisms to datagrams. The security variables can be divided into security policy variables and security association variables." [AAL96]

The security policy variables determine the enforcement of host security policy at the IP layer, and control the selection of security associations for datagrams. An organization can decide for itself on what kind of security services it uses. These decisions make up the security policy of the organization. An individual user or application may modify its security requirements within the limits of the security policy. [AAL96]

The security association variables determine the security association information between two hosts. To use the security mechanisms, the senders and receivers must agree on a key, on algorithms used, and additional parameters concerning the use of a algorithm. "This set of agreement constitutes a security association between senders and receivers" [HUI96]. "A Security Association (SA) is a relationship between two or more entities, that describes how the entities utilize security services to communicate securely. This relationship is represented by a set of information that can be considered a contract between the entities. The information must be agreed upon and shared between all the entities" [MAS95].

These Security Associations consist of security variables, that express what security mechanisms to use. A Security Association is one-way. Thus, a communication session between two hosts normally have two security associations in use, one for each direction. [I-D96a]

A Security Association include a Security Parameters Index (SPI), and source and destination addresses. The receiving host uses a combination of the SPI value and the destination address to distinguish the correct association. The sending host uses some combination of the source and destination addresses, the source and destination ports, and the user identification. [I-D96a]

3.3.2 Security Mechanisms

IPSEC offers authentication, integrity and confidentiality of IP datagrams. These services are provided by two different security mechanisms: Authentication Header (AH) and Encapsulating Security Payload (ESP). [I-D96a]

Authentication Header

The IP Authentication Header is used to provide authentication, integrity and replay protection to IP datagrams. This is done by computing a cryptographic authentication function over the IP datagram. The sender computes the authentication data and then includes the authentication data in the Authentication Header of the outgoing packet. The receiver verifies the correctness of the authentication data upon reception. [I-D96b]

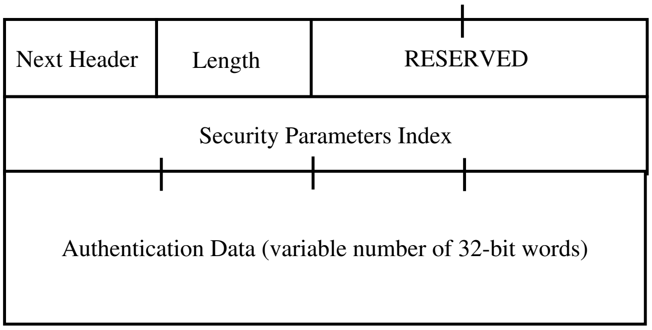

FIGURE 5. The Authentication Header Syntax

The Authentication Header syntax can be seen in Figure 5. The Next Header field identifies the next payload after the AH. The length field tells the payload length. [I-D96b]

The Security Parameters Index is the SPI value identifying a particular Security Association. The Authentication data is the actual authentication data that is calculated. At present, the MD5 and the SHA authentication algorithms must be supported. [I-D96b]

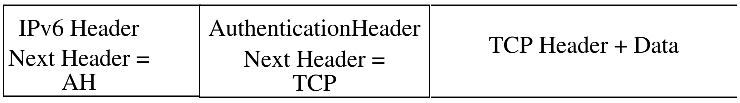

FIGURE 6. Example of IPv6 datagram containing Authentication Header

Figure 6 shows an example of an IPv6 datagram containing an Authentication Header. The Authentication Header is inserted between the IPv6 Header and the upper protocol data. In IPv6, the Authentication Header normally appears after the IPv6 Hop-by-Hop Header and Fragmentation Header but before the Destination Options Header. Fragmentation always occur after AH processing and reassembly occurs before AH processing, so if a Fragment Header exists in the packet, the Authentication Header must not precede the Fragmentation Header. [I-D96b]

Encapsulating Security Payload

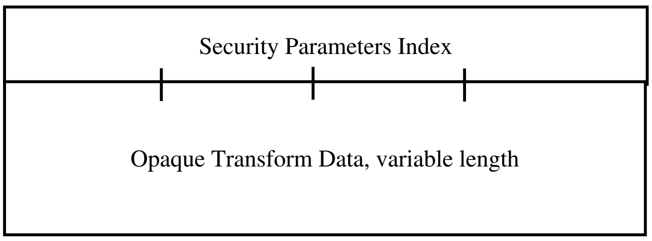

FIGURE 7. The Encapsulated Security Payload Header Syntax

The Encapsulating Security Payload (ESP) is designed to provide integrity, authentication and confidentiality to IP datagrams. ESP uses MD5 and SHA in combination with DES or 3DES. [I-D96a]

Figure 7 shows the syntax of the ESP header. It consists of the 32 bit Security Parameters Index and Opaque Transform Data of variable length. The Transform data include the encrypted payload data and the authentication data. [I-D96e]

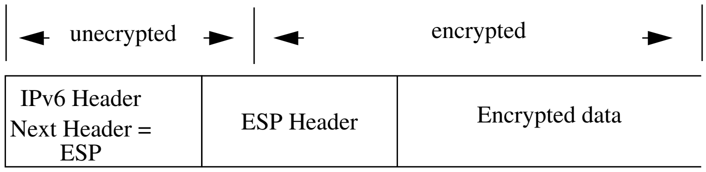

FIGURE 8. Example of IPv6 datagram containing Encapsulated Security Payload.

Figure 8 shows an example of an IPv6 datagram containing ESP. The ESP may appear anywhere after the IP header and before the final transport layer protocol. [I-D96e]

ESP can be used in two modes, Transport Mode and Tunnel Mode. Transport Mode encapsulates a transport layer frame, for example a TCP datagram, inside ESP. Tunnel Mode encapsulated an entire IP datagram inside ESP [I-D96e]. "Tunnel Mode is always used by security gateways for all packets not originating on the gateway. Tunnel mode can be used to create virtual private networks between sites protected by security gateways implementing ESP" [I-D96a]. The idea is that security gateways can be used to hide information of the network behind the security gateway. Thereby, organizations can communicate securely between two security gateways.

Combining Security Mechanisms

IPSEC offers authentication, integrity and confidentiality. All security transformations do not, however, offer all these services. In order to get all security services, combinations of AH and ESP are allowed. The following sequences of combinations of AH and ESP must be supported by an IPSEC compliant host: AH, ESP (tunnel), ESP (transport), AH-ESP (transport), AH-ESP (tunnel), ESP(tunnel)-AH, AH-ESP(tunnel)-ESP (transport), and ESP(tunnel)-ESP (transport). IPSEC-compliant security gateways must support: AH, ESP (tunnel), and AH-ESP (tunnel). [I-D96a]

3.3.3 Security Control

The control of the security mechanisms is based on the security variables. Decisions about whether security mechanism should be applied are made by control. Based on the security associations, the appropriate transformation functions are called. The control also takes care of the security failure processing. If a valid security association does not exist for a received datagram, or if the authentication and integrity checks fail, or the decryption fails, the datagram must be discarded. The failure must also be logged in some kind of system log. [I-D96e]

3.3.4 Security Management

"Security management is separated from the IP security mechanisms by the security variable interface. Security management is responsible for establishing and updating security policy variables and security associations" [AAL96]

The security management protocol that will be used with IP layer security is not specified in the IPSEC drafts. This leaves the management issues for the designer to implement. Thereby, different security management systems can be implemented, that are conformant with the IPSEC specifications. The concept of security association separates the management from the security mechanisms. The former establishes and updates the variables, while the latter read and use those values. The management is coupled to the mechanisms only via the Security Parameters Index. [I-D96a]

3.4 Key Management

All the security of an cryptographic algorithm lies in the key. [SCH96] If the key can be broken or if it is compromised, the key cracker can read all your encrypted messages. Therefore, the security of key management is a very important part of protocol security. The safest way of distributing keys between parties is for the parties to meet in private. This is however not feasible when one has to use many different keys or when one has to change keys frequently. As this is the case in data communication, some means of automatic key exchange must be used.

The solution to the key distribution problems is to use key management protocols. Concerning IPSEC, there is currently a draft for taking care of the key management. The protocol is known as Internet Security Association and Key Management Protocol (ISAKMP) [I-D97].

ISAKMP defines procedures and packet formats to establish, negotiate, modify and delete Security Associations. It defines payloads for exchanging key generation and authentication data. These formats provide a consistent framework for transferring key and authentication data which is independent of the key generation technique, encryption algorithm and authentication mechanism. [I-D97]

ISAKMP uses public key cryptography for supporting key establishment and key generation. ISAKMP provides authentication of key exchange. It also provides back traffic protection, which means that previous traffic is protected even if keys used for protecting future traffic is compromised. [I-D97]

ISAKMP is distinct from the key exchange protocols. This aims to separate the details of security association management from the details of key exchange. Thus, there may be many different key exchange protocols, each with different security properties. ISAKMP serves as the common framework for agreeing to the format of SA attributes, and for negotiating, modifying and deleting SAs. [I-D97]

One such key establishment protocol is the Oakley key determination protocol [I-D97]. Oakley uses Diffie-Hellman as the key exchange method to create session keys on hosts. Oakley is classified as mandatory in the ISAKMP draft. [ID-97]

For more information on key management refer to ISAKMP and Oakley drafts and to the master's thesis of Kaj Höglund. [HÖG97]

4 Object-Oriented Protocol Design

The OSI (Open System Interconnection) model is a model for building data communications services. It essentially divides the data communication task into several different parts, called layers. Each layer takes care of a particular part of the data communication service. Each layer provides services to the upper layer and requests services from the lower layer.

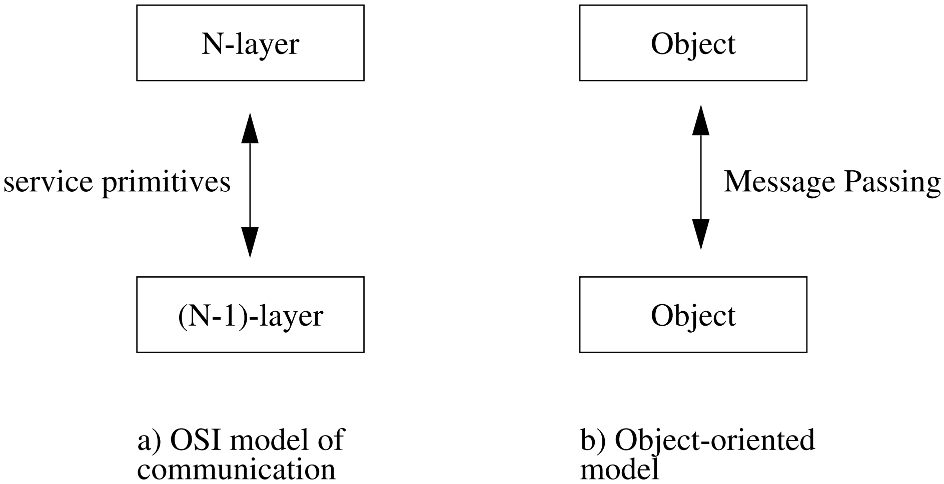

FIGURE 9. Comparison between the OSI model and object-oriented model

Figure 9 shows a comparison between the OSI model of communication and object-oriented model of communication. In the OSI model, the layers are communicating through primitives defined between the layers. The N-layer entity requests services of the (N-1) layer. Thus, the (N-1) layer provides services to upper layers. In the object-oriented world, the objects are communicating with each other through message passing. "Action is initiated in object-oriented programming by the transmission of a message to an object responsible for the action. Message passing is the process of locating and executing a method in response to a message" [BUD96]. The messages may also be objects passed between the communicating objects.

Thinking object-oriented, the layers of a protocol stack could be seen as objects communicating with each other. This would motivate the use of object-oriented methods when designing protocol software. [MAR97]

4.1 Design Patterns

"Designing object-oriented software is hard, and designing reusable software is even harder" [GHJV95]. It takes a long time for a novice to learn object-oriented programming and thinking. Often, novice programmers tend to fall back on non-object-oriented techniques they used before.

Experienced programmers tend to reuse solutions that have worked for them in the past, not solve every problem from scratch. Recurring patterns of classes and communication objects can be found and used again and again. [GHJV95]

[GHJV95] defines the term design pattern as follows:

"A design pattern systemically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context".

Gamma et. al. have categorized different design patterns and collected the most common design patterns into the book Design Patterns. The patterns are divided into three different patterns classes: Creational Patterns, Structural Patterns and Behavioral Patterns.

4.1.1 Creational Patterns

Creational Patterns abstracts the instantiation process. They help make a system independent of how its objects are created, composed and represented. The patterns encapsulate knowledge about which concrete classes the system uses. They hide how instances of these classes are created and put together. [GHJV95]

The Singleton Pattern is an example of a Creational Pattern. It ensures that a class has only one instance, and provides a global point of access to it. Figure 10 shows the Singleton Pattern structure. The Singleton class itself is responsible for keeping track of its sole instance. [GHJV95]

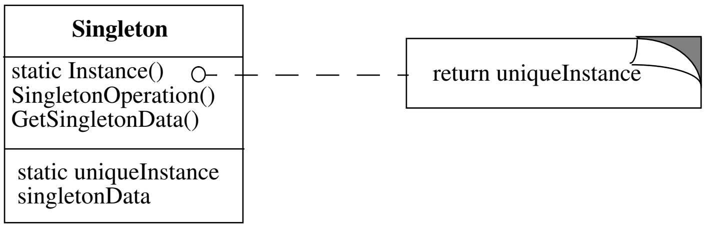

FIGURE 10. The Singleton Pattern Structure [GHJV95]

4.1.2 Structural Patterns

Structural Patterns are concerned with how classes and objects are composed to form larger structures. The Composite Pattern shown in Figure 11 is a Structural Pattern. With the Composite Pattern, one wants clients to be able to ignore the difference between compositions of objects and individual objects. Clients treat all objects in the composite structure uniformly. For example, a client may want to draw different kinds of graphical objects, e.g. a line or a rectangle. Having to distinguish these objects, makes the application more complex. When using the Structural pattern, the client does not have to concern about what kind of element it wants for example remove. It just calls the remove operation of superclass, and the right operation is invoked. [GHJV95]

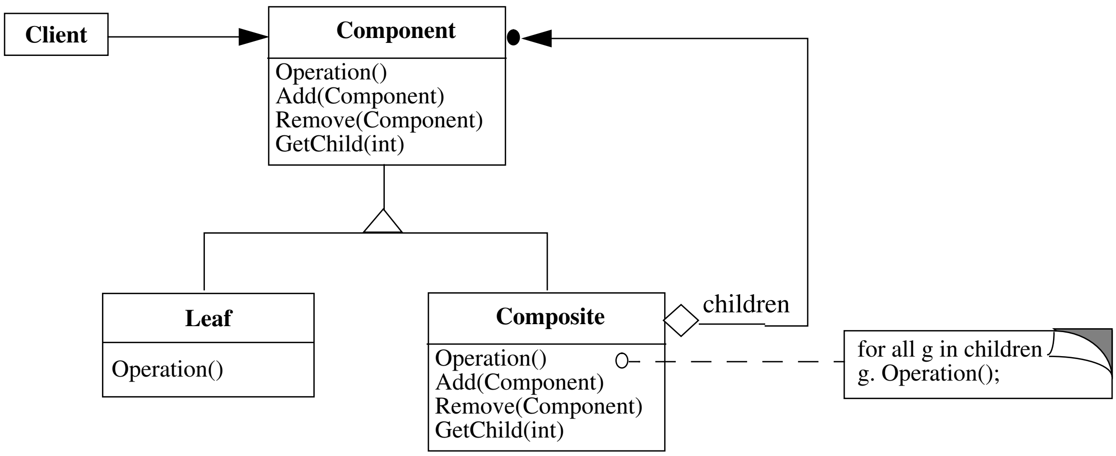

FIGURE 11. The Composite Patterns Structure [GHJV95]

4.1.3 Behavioral Patterns

Behavioral Patterns are concerned with algorithms and the assignment of responsibilities between objects. These patterns also describe the patterns of communication between objects and classes. Behavioral object patterns use object composition rather than inheritance.

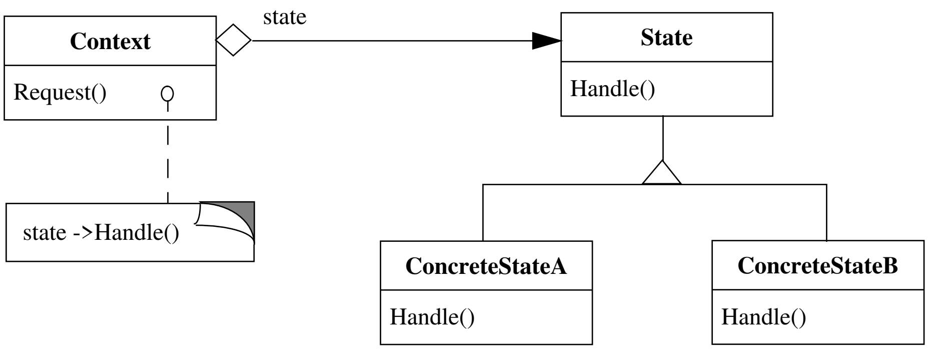

FIGURE 12. The State Pattern Structure [GHJV95]

Figure 12 shows the State Pattern. The key idea in this pattern is to introduce an abstract State class, that has concrete State classes as subclasses. The actual object, for example a protocol object, maintains an instance of a subclass that represents the current state of the object. The protocol object delegates all state-specific requests to this state object. Whenever the protocol object changes state, the object changes the state object it uses. [GHJV95]

4.2 Conduits+ framework

"A framework embodies a generic design, comprised of a set of cooperating classes, which can be adapted to a variety of specific problems within a given domain" [CP95]. In the traditional procedural programming, the control in the program is implemented by a programmer. When using a framework, however, most of the control is the responsibility of the framework. Thus, the framework parts can be reused, and the programmer can concentrate on the environment. This approach is also taken in the Conduits+ framework. This chapter first considers the benefits of using a framework, and then the Conduits+ framework is presented.

4.2.1 Benefits of Using a Framework

Frameworks return a number of benefits to the developers. Some of them are attributable to the reuse of code, others are unique to frameworks. A list of benefits is as follows: [TAL95]

- **Less code to design and implement**. By providing the infrastructure, the framework decreases the amount of standard software that one must design, code, test and debug.
- **More focus on areas of expertise**. When using a framework, you can focus on the area where one can add the most value to one's code. Frameworks create an environment in which solving domain problems, not programming problems, is possible.
- **Proliferation of expertise.** Frameworks allow organizations to package the common characteristics of their expertise and makes it possible to resell specialized knowledge.
- **Improved consistency.** Because frameworks embody expertise, you solve the problems once, and you can use the designs captured in the framework consistently across all problems in the frameworks domain.
- **Generic solutions that can be reused for other, related problems.** Programmers who are used to dealing with frameworks tend to think in terms of generic solution rather than special solutions.
- **Improved integration of related programs**. All programs based on a particular framework inherit common design. Therefore, programmers who are working on similar programs have a better chance of understanding and working with each other's code. The programs thus more likely work together consistently an reliably.

- **Reduced maintenance overhead.** Because your applications are based on a framework, generally any change in the framework automatically updates in the applications.

A protocol stack is usually implemented with a layered software. This does not address reusability of across layers very well. Herman Huni,Ralph Johnson and Robert Engel have been developing a framework called Conduits+ for writing efficient, reliable, flexible and reusable network protocol software. On this basis a TCP/IP implementation and an implementation of the signalling system of a multi-protocol ATM access switch has been developed [EHJ96].

The primary goal of the framework is to support implementation of reusable object-oriented software. This is done by offering a framework relying heavily on Design Patterns [EHJ96].

The framework is made up of two sorts of objects: namely conduits and information chunks. Information chunks are everything flowing through the conduits. A conduit is a software component with two distinct sides. These sides may be used to connect a conduit to another, neighbour conduit. [EHJ96]

4.2.2 Conduit types [EHJ96]

There are four kinds of different conduits, a Protocol, a Mux, an Adapter and a ConduitFactory.

Adapter

An Adapter is used to interface the Conduit framework to some other software or hardware. Thus, only its side A is connected to another conduit. In the case of IPv6, an Ethernet adapter is needed. This adapter is connected to Ethernet adapter code written in C. The interface could also be for example a socket interface to some application using the protocol stack built with Conduits+.

Protocol

A communication protocol can be described by a finite state machine (FSM). The Protocol conduit is used to implement this state machine of the protocol. In these conduits, information chunks are produced, consumed and tested. The protocol remembers the current state of the communication. The protocol conduit has exactly one neighbour conduit connected to both of its sides.

Mux

A Mux is a conduit that connects one sideA conduit to any number of sideB conduits. Thus, the Mux takes care of multiplexing the connections. The sideB instance can for example be realized through a List. The Mux must be able to extract some kind of key from an information chunk in order to determine to which Protocol the chunk has to be routed.

ConduitFactory

The ConduitFactory takes care of adding new Protocol conduits to a Mux. It also takes care of the case when there is no existing conduit on sideB that could handle a particular information chunk. When such a situation occurs, the information chunk is routed to the ConduitFactory. The ConduitFactory then installs a new Protocol conduit that can handle the chunk. The ConduitFactory is usually connected to two Muxes.

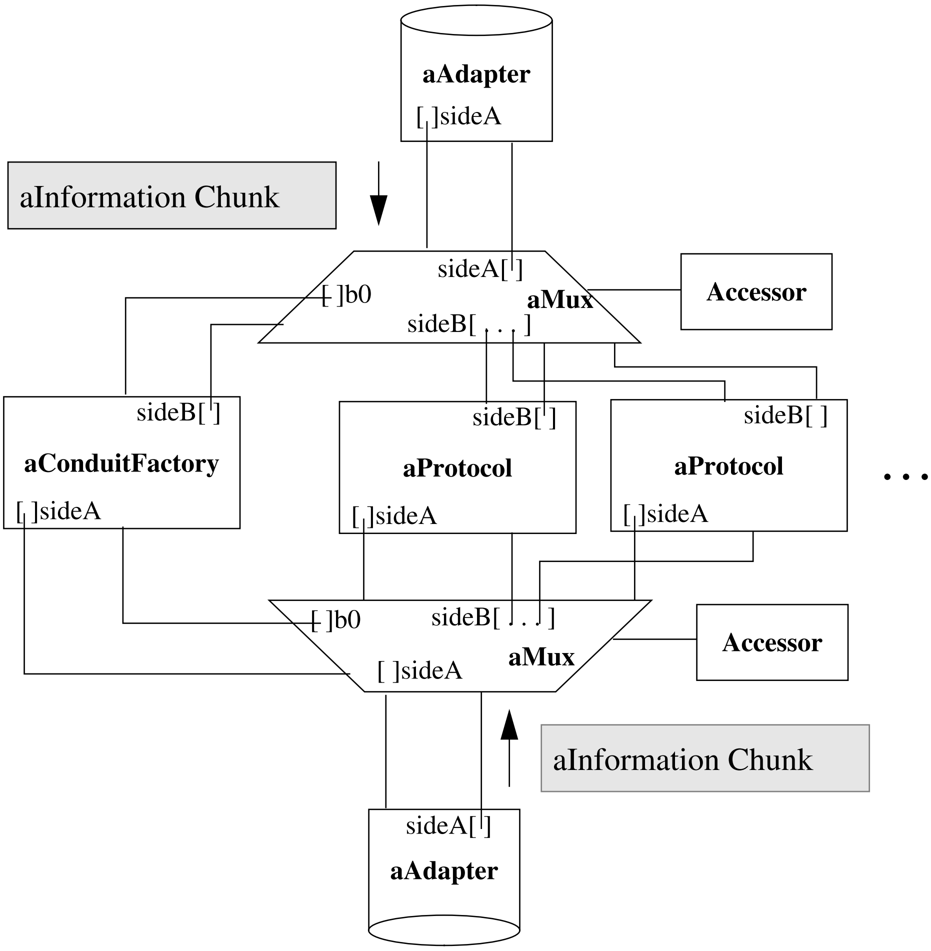

FIGURE 13. An example Conduits+ graph

Figure 13 shows an example Conduits+ graph. The two Adapters are used to connect the Conduit entity to the outer world. There are two Muxes, one that routes incoming packets and one that routes outgoing packets. Between the Muxes there are a number of Protocol conduits. New Protocol conduits can be added by the ConduitFactory.

4.2.3 The Use of Design Patterns

The architecture described is not as reusable and extendible as it should be. The architecture can be improved by using existing

Design Patterns, resulting in a black-box framework that is elegant, highly reusable and easily extendible.

To reuse the same Mux class, components called Accessors, are used. These access the relevant dispatch key within an information chunk. In other words, the Accessors takes care of the environment dependent parts of the Mux. It computes the index of the sideB conduit to which the current Information Chunk should be dispatched. It also computes an index for a new sideB conduit. To achieve this goal, the Strategy Pattern is used. The strategy Pattern separates the algorithm from the object. [EHJ96] [GHJV95]

The Protocols are implemented using the State Pattern. Each State of the Protocol is represented by a separate object. To create the different States, the Singleton Pattern is used. The singleton pattern ensures that only one instance of every State class is created and used. Thereby, every Protocol uses the same State object. The protocols just have references to the State objects. [EHJ96] [GHJV95]

The messenger transporting the information chunks is developed with the Visitor Pattern. The intent of this pattern is to represent an operation to be performed on the elements of an object structure. Visitor lets one define a new operation on conduits without changing the classes of the conduits. The conduits only have to deal with Visitors. When new operations are needed, one creates a new subclass of the Visitor class. [EHJ96] [GHJV95]

4.3 The Java Language

Originally, Java was planned as a programming language for software for consumer electronics. The team planning the software worked at Sun Microsystems and was headed by James Gosling.

[GM96] describes java as follows:

Java: A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multi-threaded, and dynamic language.

Java is meant to be a simple language. In order to keep the language both small and familiar, the language is based on C and C++, but the designer removed a number of features. For example, Java does not support the goto statement and there are no pointers. [FLA96]

Java is an object-oriented language. "Simply stated, object-oriented design is a technique that focuses design on the data (=objects) and on the interfaces to it. To make an analogy with carpentry, an "object-oriented" carpenter would be mostly concerned with the chair he was building, and secondarily with the tools used to make it; a "non-object-oriented" carpenter would think primarily of his tools". [GM96]

Java is designed to support applications on networks, it is a distributed language. Java has library routines for coping with TCP/IP protocols like HTTP and FTP. Java supports reliable stream network connections with the Socket class.

Java is a language that is interpreted. The Java interpreter can execute Java bytecodes directly on any machine to which the interpreter has been ported. [GM96]

Java is a robust language. Its origin as a language for software for consumer electronics means that Java has been designed fro writing highly reliable software. Java puts a lot of emphasis on early checking for possible problems, later dynamic (runtime) checking, and eliminating situations that are error prone.

Java is meant to be a secure language. Java does not have pointers, so a programmer can not forge pointers to memory. The Java compiler does not handle memory layout decision, so a programmer can not guess the actual memory layout of a class by looking at its declaration. Security is of course not easy to maintain, but Java is designed to make rogue software difficult to write. [FLA96]

Java is architecture neutral. The compiler generates byte code rather than native machine code. The interpreter and run-time system implement a virtual machine called the Java Virtual Machine. Thus, Java programs can be run on any system implementing the virtual machine [FLA96].

Java is a portable language. Being architecture neutral is a big part of being portable. But the tight typing makes it even more portable. That is because the types are always the same size, for example an int is always a signed two's complement 32 bit integer. In addition, the Java compiler is written in Java, while the Java run-time system is written in ANSI C. [FLA96]

Java is an interpreted language and is never going to be as fast as a compiled language. But to support high performance, the byte code was designed with "just in time" compilers in mind. These compilers can translate Java byte codes into machine code for a particular CPU at run-time.

Java is a multithreaded language. It provides support for multiple threads of execution, that can handle different tasks. One benefit of multithreading is that it improves the interactive performance of graphical applications for the user. Java provides a built-in language support for threads. This includes a set of synchronization primitives. The threads are a good aid in designing protocol software. Java is a dynamic language. Java loads in classes as they are needed, even across a network. [FLA96]

The portability, the support for threads and network programming, the garbage collection facilities and the dynamic run-time behaviour makes Java a convenient tool to be combined with the Conduits+ framework.

4.3.1 Inheritance in Java

The two most common techniques for reusing functionality in object-oriented system is class inheritance and object composition. Class inheritance lets one define the implementation of one class in terms of another's. Reuse by subclassing is referred to by white-box reuse. The white box refers to visibility. The internals of parent classes are often visible to subclasses. [GHJV95]

In object composition, new functionality is obtained by assembling objects to get more complex functionality. This requires that the objects being composed have well-defined interfaces. This style of reuse is called black-box reuse, because no internal details of objects are visible. [GHJV95]

Multiple inheritance means that a class can inherit from several other classes. This may cause problems like for example diamond inheritance. It means that the inheritance is ambiguous if for example a method is inherited from both the superclasses. It is not therefore clear, which method the subclass should use in different situations. [AG96]

Multiple inheritance is not supported by Java. A new class can only extend one superclass. Extending a class means that the new class inherits the superclass's implementation One can declare classes abstract, which means declaring classes that define only part of an implementation. Extended classes can then provide specific implementation of some or all of the methods. [AG96]

The problems of multiple inheritance arise from the multiple inheritance of the implementation. To avoid this problem, Java provides a way to inherit the contract, namely the method declarations, without inheriting the implementation. The way is to declare an interface type instead of a class type. A class can then inherit from multiple interfaces. [AG96] Interfaces are used in the Java Conduits+ framework for offering a more flexible and generic solution.

5 Design and Implementation

This chapter presents the design and implementation of the IPv6-IPSEC prototype. Main emphasis is put on the design of the Conduits+ Architecture. The design choices are presented and motivated. The class diagrams of the implementation are shown. The chapter concludes with diagrams of the prototype functionality.

5.1 Implementation Architecture

The implementation consist of the Conduits+ framework, the IPv6-IPSEC implementation and the Ethernet adapter. The prototype is implemented in Sun Solaris 2.5. The Conduits+ framework and the IPv6-IPSEC parts are implemented in Java using the Java Developers Kit (JDK). The Ethernet adapter is implemented in C. The Conduits+ framework is based on the original Conduits+ ideas and some new ideas by M. Sc. Pekka Nikander.

Figure 14 shows a conceptual model of the IPSEC-IPv6 implementation. IPSEC is incorporated into the IPv6 prototype. The security control and the security mechanisms are implemented as part of IPv6. The security variables are the interface to the Security Management. The Security Management updates and maintains the variables. The security policy decides which kind of security scheme is maintained. The Security Management is responsible for the policy. Therefore some User interface has to be supplied to the Security Management, so that the security policy decisions can be configured. However, in this implementation the security management is done through manual configuration. The IPSEC prototype is built based on the Internet Drafts from 1996. Currently, new Drafts are being published, so the prototype implementation is out of date.

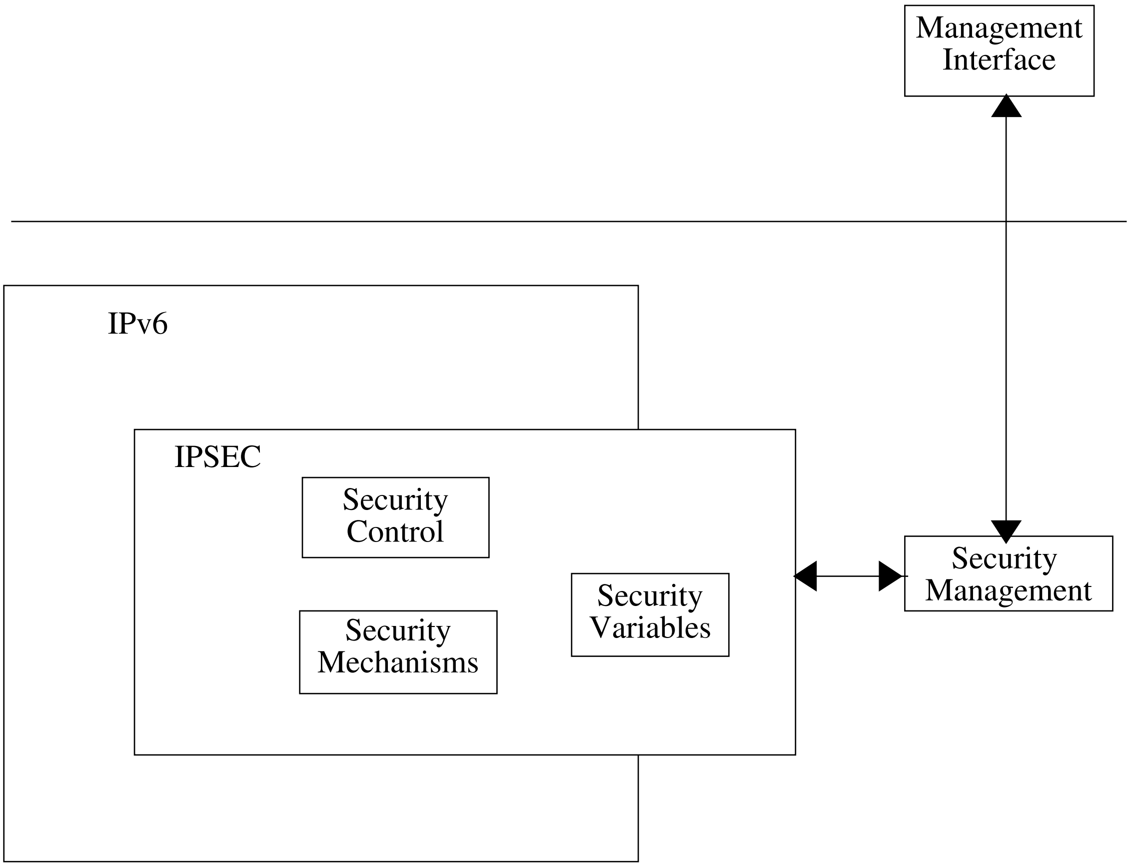

FIGURE 14. A conceptual model of the implementation

The Ethernet driver /dev/le is a UNIX STREAMS hardware device driver. It is used through the DLPI interface. Ethernet packets can be received and sent using the Ethernet driver. The Ethernet adapter is the interface to the actual Ethernet. It opens and closes the Ethernet driver, and sends and receives Ethernet packets.

Figure 15 shows the test environment. Two hosts are in use, and the packets are sent between them. The IPv6 packets are created in one end and security transformations are performed based on the manually configured security policy. Then, the Ethernet packet are forwarded to the Ethernet adapter, and sent through Ethernet to the receiving host.

The system consists of three threads. The first one waits for packets to be received at the Ethernet adapter, the second one waits for data to be received at the upper adapter, and the third one transports the visitors through the Conduits+ graph.

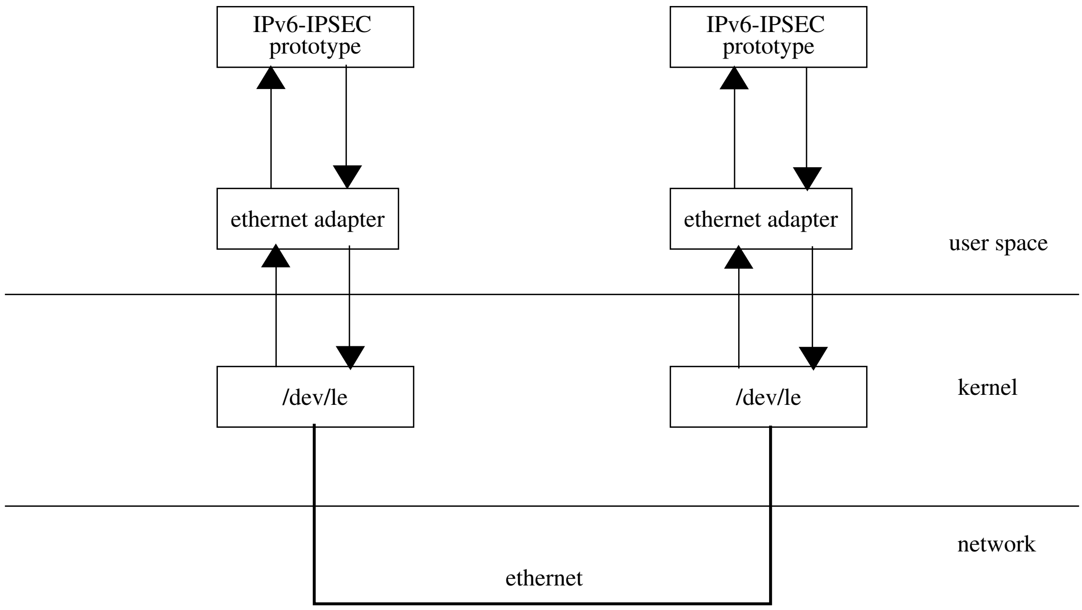

FIGURE 15. The test environment

5.2 Conduits+ Architecture

The IPv6-IPSEC Conduits+ architecture is presented in Figure 16. The main idea behind the architecture is to incorporate the IPv6 philosophy into the architecture. As little processing as possible should be done to the regular IPv6 packets. Special processing, for example security transforms, is deferred to option headers that can be dynamically added after the actual IPv6 header. Therefore, the number of options can not be exactly known when receiving a packet.

This means that the architecture should be dynamic, so that new headers can dynamically be added when building the packet. Also, headers should be dynamically handled when receiving a packet. This implementation suggests using a loop in the Conduits+ graph for processing packets dynamically. This functionality is not implemented yet.

The loop is implemented with the aid of four muxes. The inner muxes of the figure take care of routing the datagram to the right module. The decisions on which module the datagrams should be routed to are made based on the security policy. The outer muxes take care of routing the packets through the loop. When all the headers have been processed, the outer muxes route the datagram forward in the graph: to the upper adapter for received datagrams and to ipCheckHdrSession for datagrams to be sent

The figure shows three header handling modules: ipmodule, espmodule and ahmodule. All IPv6 options headers require the same kind of module. Regular IPv6 packets are sent through the ipmodule.

The upper adapter is an adapter interfacing IPv6 to upper layer protocols. For testing purposes, in this implementation the adapter reads packets from a file and sends them over to the other end, where the packets are likewise saved in a file.

The ethernetAdapter receives and sends packets through the network. The interface to the Ethernet driver consists of four functions. The open() function opens the Ethernet driver and the close() function closes the driver. receivePacket and sendPacket are used to receive and send datagrams through the Ethernet.

The ipForwardSession checks if the IP datagrams have reached there destination. If not, the datagrams should be forwarded. In this imple-

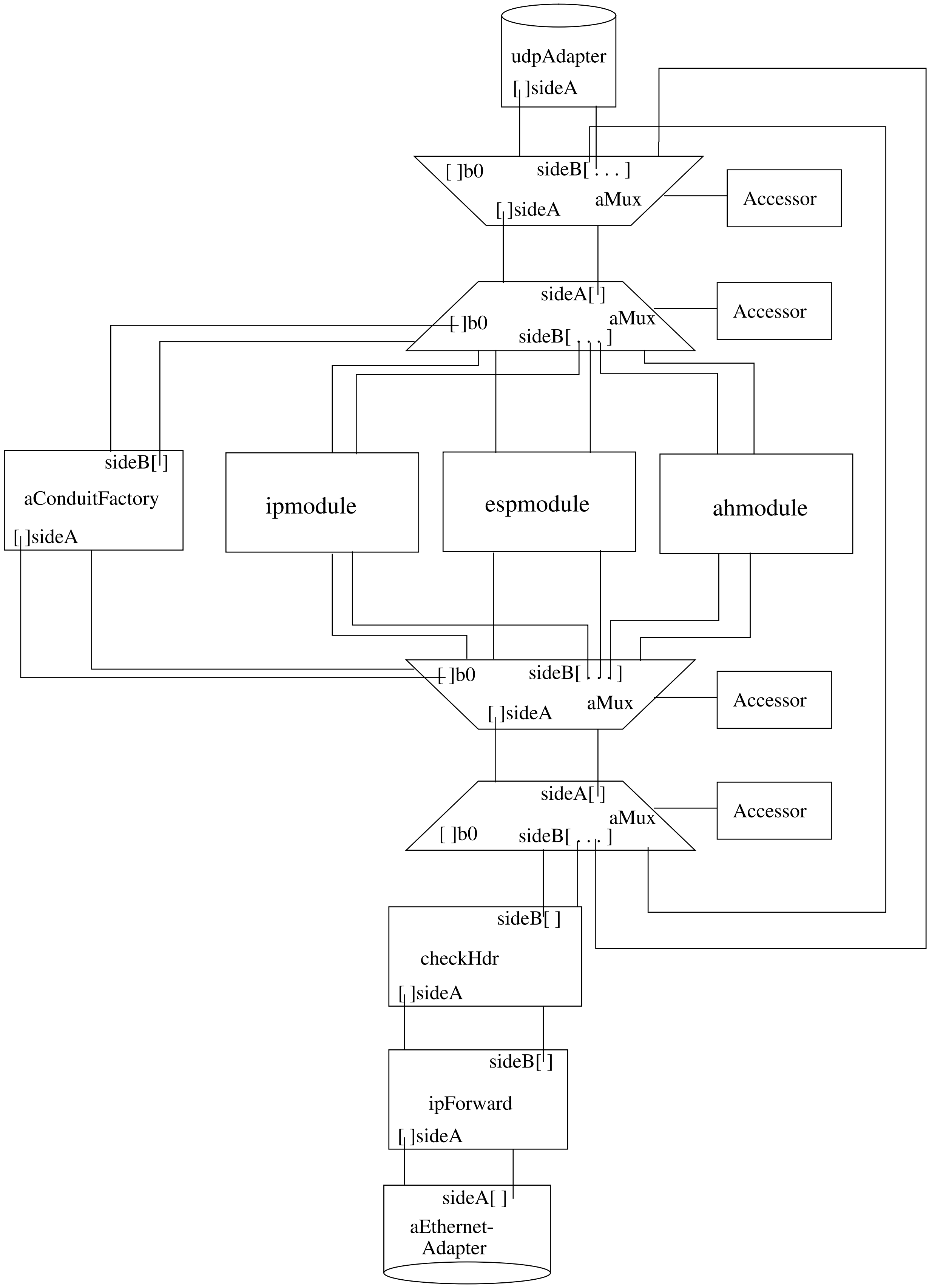

FIGURE 16. The ipsec arc hitecture

mentation, however, the datagrams are dropped. The ipCheckHdr session checks the consistency of received IP datagrams.

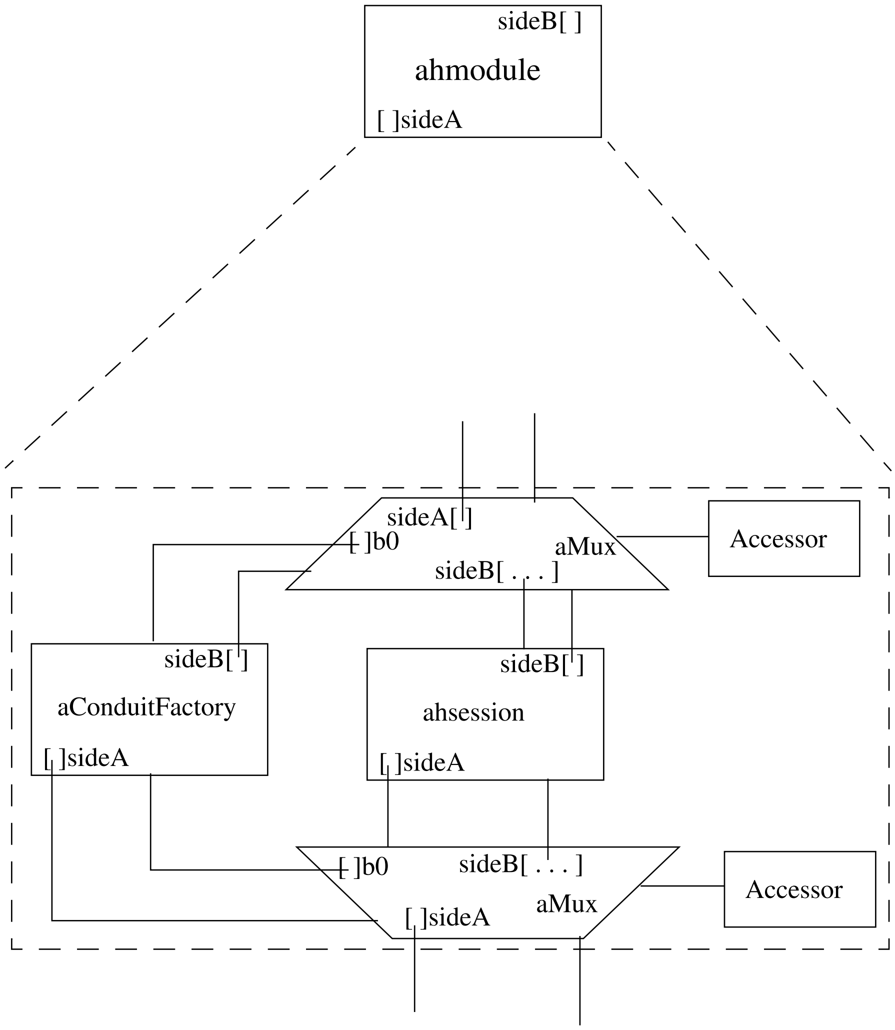

FIGURE 17. The Authentication Header Module

Figure 17 shows an example of a header module, the Authentication Header module. All the other modules have the same architecture. The muxes routes the packets to the corresponding session according to security association information. Initially, no sessions exist between the modules. When a new session must be created, the

ConduitFactory sends an installer attach the sides of the Session to the muxes. The sessions each takes care of the packets that are related to a certain security association. If the datagram authentication is not valid or no corresponding security association exist, the datagrams are dropped.

5.3 Class Model of the Implementation

The class model of the implementation is shown in Figures 18-20. Figure 18 shows the Conduits+ framework class model. As can be noted, this model differs slightly from the original Conduits+ framework model. In this framework, the Protocol is renamed and is called Session. Therefore, the framework parts are Adapter, Session, Mux and ConduitFactory.

The implementation includes two Adapters, an Ethernet adapter and an upper adapter. The responsibility of the Ethernet adapter is to transmit packets. The actual transmission is done with the part of the Ethernet adapter that is written in native code.

The upper Adapter reads packets from a file and creates MessageTransporters that are put on the upper Adapter queue. The MessageTransporters are fetched by the ControlThread. Then the MessageTransporters are moving through the Conduits+ graph and sent through the ether.

The Session is an abstract class, which all the actual Session inherit. The Session classes include methods for binding a side to another Conduit and for accepting a Visitor coming from another Conduit. The Session itself does no processing on the actual packets. The processing is done in the States. Each Session has a reference to a State object. The Session also has a method, by which it can change the reference to another state. In this way, the State can be changed. This implementation includes only one State per Session. This is because the IPv6 protocol is stateless.

The Mux entity is an interface. Its purpose is to declare the contract specific for the Multiplexor Conduit. The Multiplexor entity is a concrete class. Therefore, all the Multiplexors look the same. There is two attach methods, one for attaching the A side of the Multiplexor, the other for inserting a new Conduit on the B side.

The B side Conduits of a Multiplexor are maintained in a List. The right Sessions are identified by a key. This implementation uses protocol specific data, for example SPI values, directly as the key. Some kind of a hash value would be more convenient to use.

The actual routing decisions of a Multiplexor is done by the Accessors. Therefore, each Multiplexor has an Accessor of its own, and the Accessors of course all differ in implementation.

If there is no Session to route the packet to, the Visitor is routed to a ConduitFactory. In the implementation, every stage has a different ConduitFactory. The ConduitFactories therefore return a certain type of Session with some Session dependent data already filled in. For example, the ahConduitFactory inserts the important security association information when an ahSession is created. Concerning reusability, however, the ConduitFactory should be as reusable as possible and should require only slight changes as new types of Sessions need to be added.

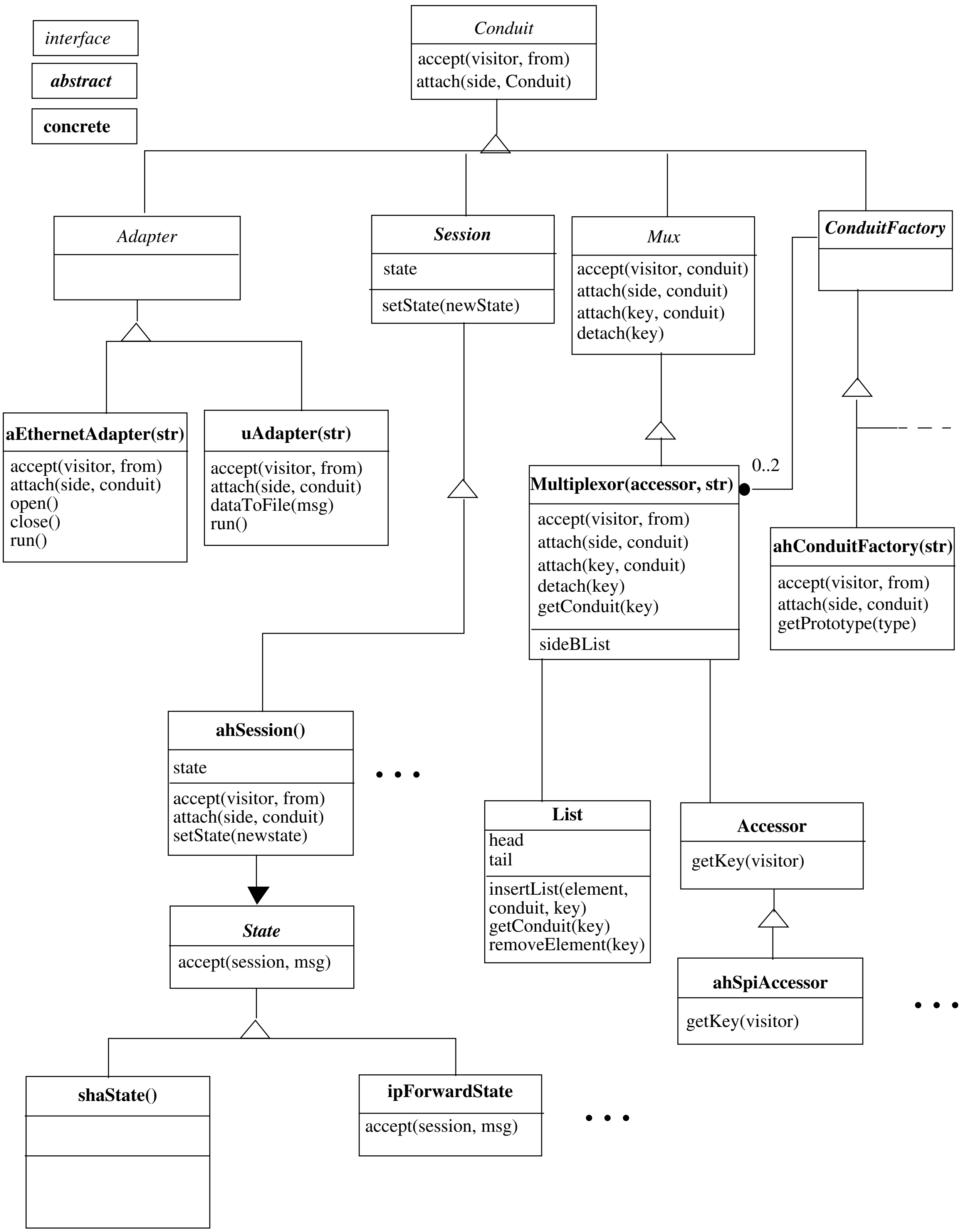

FIGURE 18. The Conduits+ framework c lass diagram

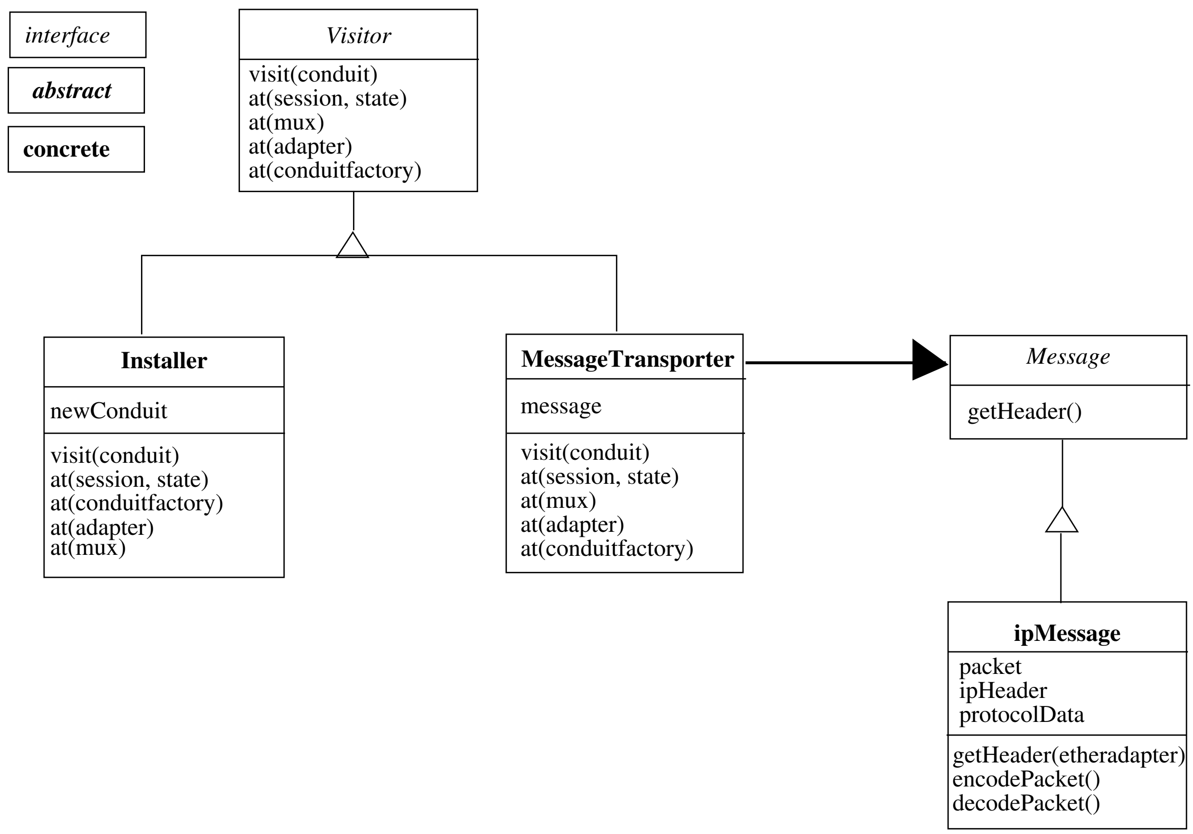

FIGURE 19. The Visitor class model

Figure 19 shows the Visitor class model. Visitors are objects that traverse the Conduits+ graph. The MessageTransporter is a Visitor that carries a message. In this case, the message is an IP Message, but the Message class can be extended to have several subclasses. The ipMessage consists of the actual datagram, which is a byte array. This is really all that is needed. An index value is held in the message, that points to the part of the byte array that should be processed next. The present implementation also includes an ipHeader and a byte array, where the actual data is copied when the packet has been processed by IP. This is included in the implementation for testing purposes.

The Installer is a Visitor, which installs new Sessions when needed. The Installer has a reference to the new conduit that should be installed. The Installer traverses the Conduits+ graph until the right Multiplexor is found and attaches the new conduit to the Multiplexor. To detach a Session, an Uninstaller would be required. This implementation does not implement this functionality.

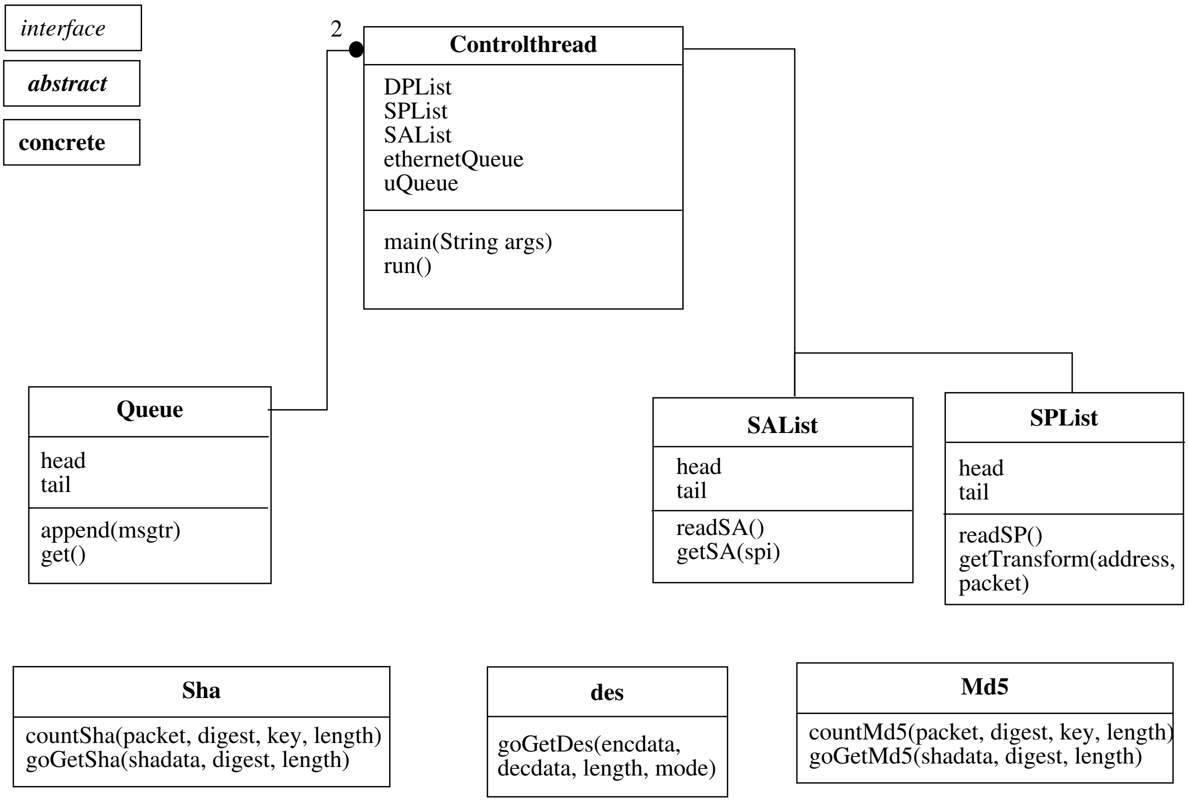

FIGURE 20. Implementation dependent classes

Figure 20 shows the implementation dependent classes. ControlThread is the main class that starts the processes and creates the initial graph configuration. It implements the thread that transports Visitors in the graph, and starts the Ethernet and upper Adapter threads. Two queues are used, one for MessageTransporters going downwards and one for MessageTransporters going upwards.

The ControlThread also uses two lists. The SPList contains the security policy information. The security policy should be decided through some kind of security policy daemon. Any such daemon is however out of scope in this implementation, so the security policy is manually configured. The other list is the Security Association list. This list contains the security associations. In this implementation, security associations are configured manually.

The prototype uses native code to do the cryptographic algorithm counting. The algorithms used are keyed SHA, keyed MD5 and DES-CBC. These are all accessed through classes of their own.

5.4 Prototype Functionality

The functionality of IPSEC is shown in the interaction diagrams below.

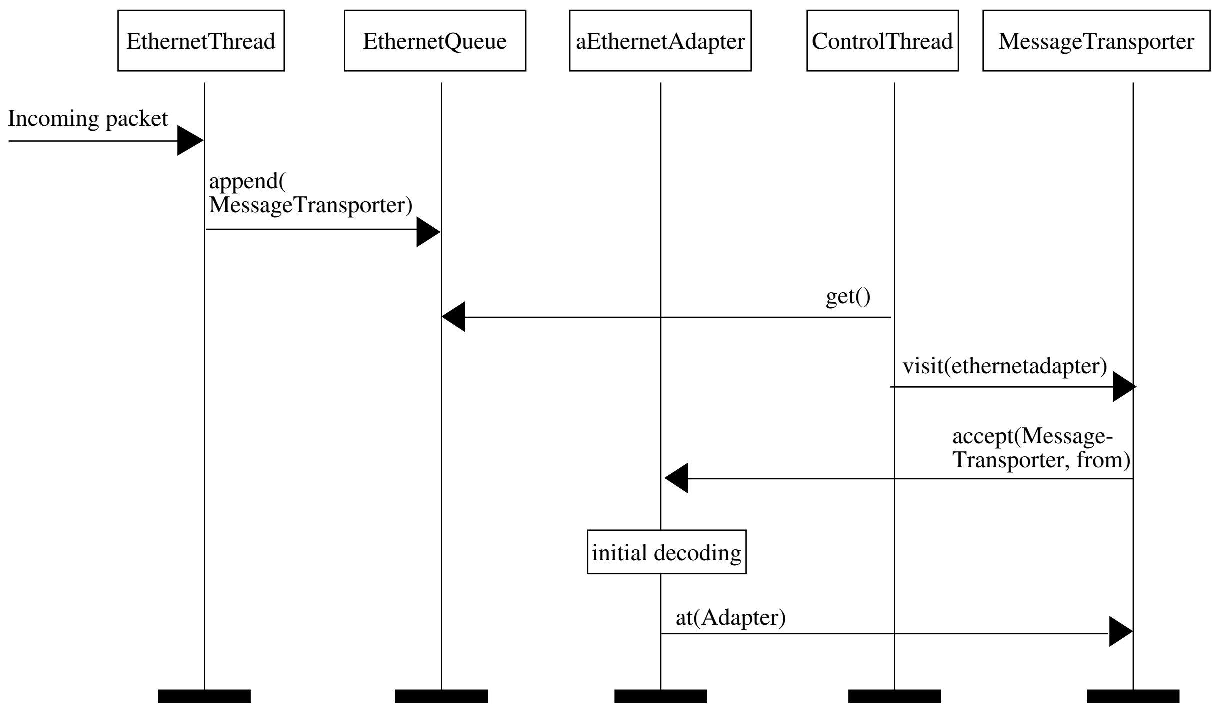

FIGURE 21. A Packet arrives from Ethernet

Figure 21 shows the situation when an incoming datagram arrives from Ethernet. The EthernetThread loops and checks for incoming packets. When the packet has been received, the EthernetAdapter creates a MessageTransporter and appends the Message to the Transporter. The Transporter is then put on the ethernet queue with the append method.

The ControlThread looks in the queue and when a MessageTransporter is put on the queue, the ControlThread removes it from the queue with the get method. Then, the ControlThread calls the visit method of the MessageTransporter. The MessageTransporter visits the Ethernet Adapter and some initial processing is done.

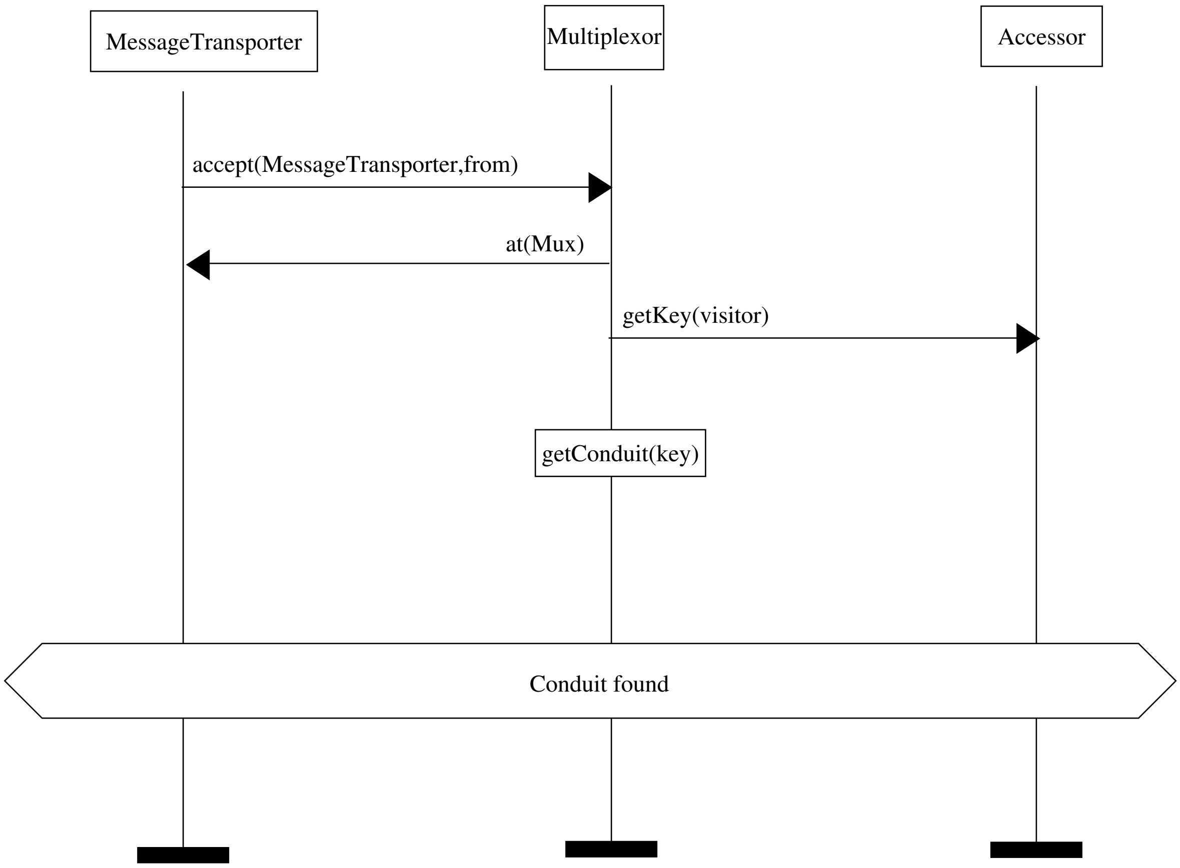

FIGURE 22. MessageTransporter arrives to a Mux

Figure 22 shows a MessageTransporter arriving to a Multiplexor. The MessageTransporter calls the Multiplexor's accept method. The Multiplexor in turn calls the method at(Mux), that tells that the MessageTransporter is at a Multiplexor. Then the Multiplexor calls the getKey method of the Accessor to get the appropriate key to a certain Session. The conduit where the MessageTransporter has to go next is found, and the MessageTransporter will visit that conduit.

Figure 23 shows the case when a new Session has to be inserted between two Multiplexors. The Accessor returns a key that does not have any Conduit attached to it. In this case, the MessageTransporter visits the ConduitFactory. Here, a new Session is created. Then, an Installer is created. The installer visits the Multiplexor and attaches the Session to the B side list of the Multiplexor. A second Installer installs the Session to the second Multiplexor. When the new Session is installed, the MessageTransporter returns to the Mux and proceeds to the Session now attached.

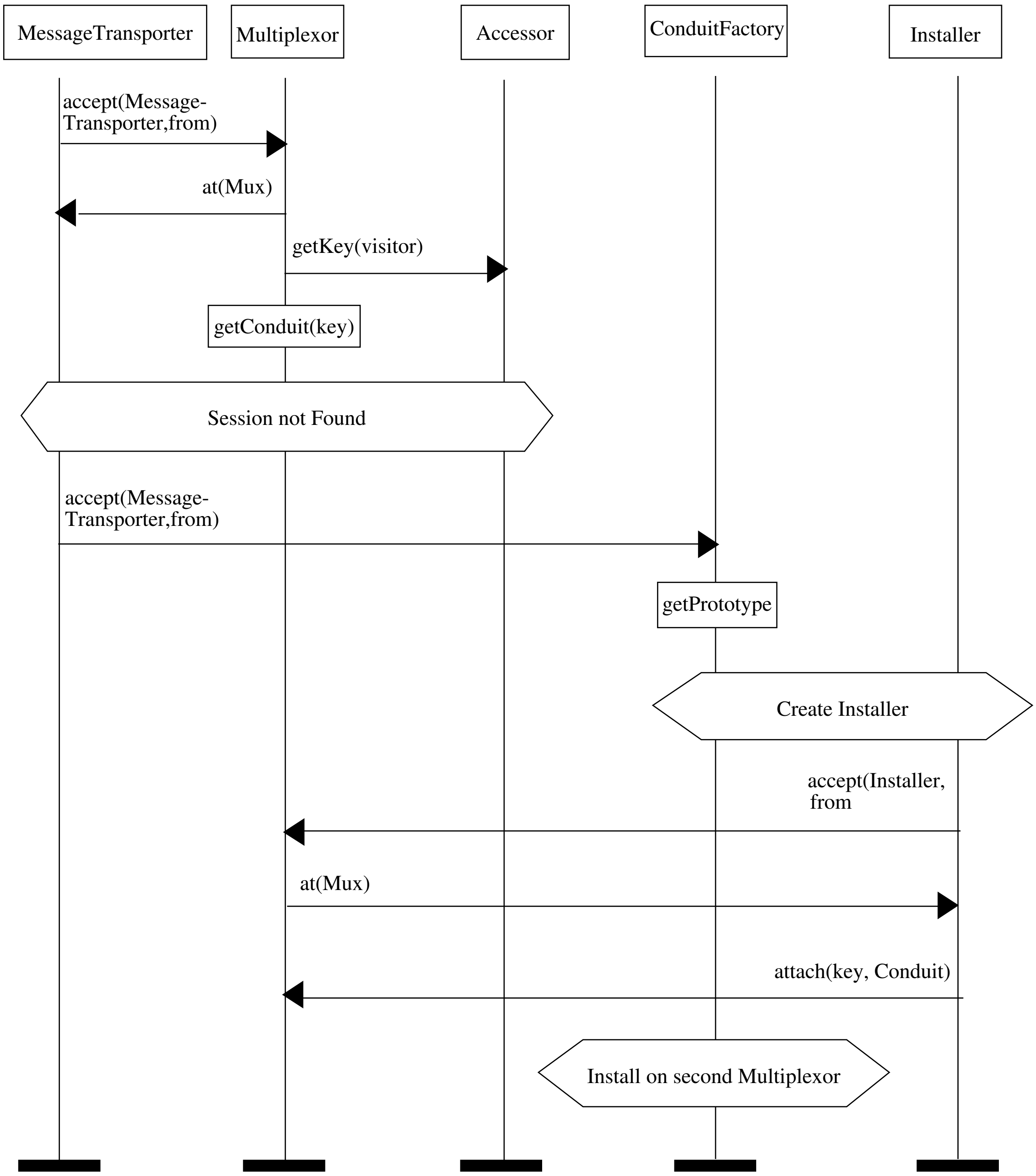

FIGURE 23. A new Session is created and inserted

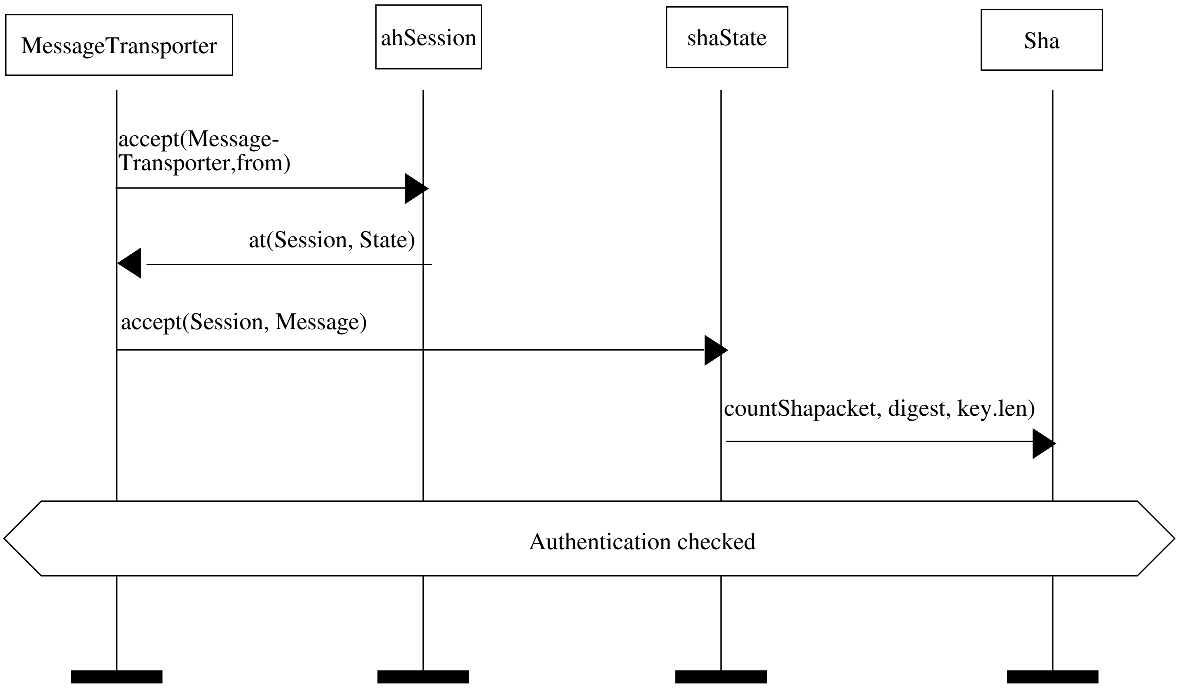

FIGURE 24. The Authentication checking procedure

Figure 24 shows the actions taken when a MessageTransporter with an incoming packet arrives to ahSession. The MessageTransporter calls the accept method of the ahSession. The ahSession then answers with calling the at(Session, State) method of the MessageTransporter. Thereby, the Session tells which state it is currently in. The MessageTransporter can then directly call the accept method of the shaState. The shaState gets the Message to be processed. The packet is sent to the Sha object and the SHA digest is counted on the packet. The authentication digest is then checked by the State. If the digests do not match, the packet is dropped. Else, the packet is forwarded for further processing.

6 Evaluation and Consideration

This chapter summarizes the results and experiences of the project. The prototype is evaluated, and IPSEC and IPv6 considerations are presented. The use of the Conduits+ framework and Java is evaluated.

6.1 Prototype Evaluation

The performance of the prototype was evaluated by some measurements of sending variable amounts of data between two SUN SparcStation 20 machines using 10Mb/s Ethernet as transmission medium. The data was first sent as ordinary IPv6 packets, and then encrypted using DES-CBC. The performance measurement values are presented in Table 1.

TABLE 1. Performance measurement values

Block size (bytes)	Ordinary IP (bytes/sec)	Encrypted IP (bytes/sec)
1	14	7
10	135	69
100	1333	578
1000	11765	3571
10000	39526	14706
100000	91659	36928
1000000	149589	42876

As can be seen from the table, 150 kb/sec unencrypted and 40kb/sec encrypted transfer speed can be achieved with the current prototype. These initial and unoptimized performance results seem reasonable when considering the achieved encrypted transfer rates of about 200 kb/sec of the Solaris kernel based IPSEC implementation in [AAL96]. With JIT compilation and optimizations it should be possible to achieve better performance values.

6.2 IPSEC and IPv6 Considerations

As mentioned before, the IPSEC specifications are Internet drafts and work is under progress. New drafts have already been released during the project. This implementation, however, is based on the drafts listed in the references, so it is out of date.

Some concern can be raised about the IPSEC specification progress. There are security problems that must be solved immediately. Right now, users solve their security problems by using application level security, such as PGP for securing e-mail and SSH for securing remote sessions and for secure file transfer. Application layer security does not however offer security against all security threats. Thus, there is a need for the stabilization of specifications and for public domain and commercial IPSEC implementations.

There is also a need for more IPv6 implementations. The philosophy of making ordinary IP packets simple and requiring minimal processing, is a strong benefit of IPv6.

6.3 Conduits+ experiences

The Conduits+ seems to be a good concept, and together with Java it seems to capture the flexible and dynamic parts of IPv6 well.

Reusable software is hard to build. The framework implemented in this project is not as reusable as it could be. Right now, the ConduitFactories create specific Sessions. The ConduitFactory should be changed to be a general part that can be reused, and when adding a new type of Session, only minimal changes would be required to the Factory. One should strive to build a framework, where changes are restricted to building new Sessions, States, Accessors and Visitors. The rest should, in my opinion, be built as reusable as possible.

One should in my opinion try to break up the protocol functionality in relatively small and simple parts. This improves the reusability, as the solutions can be mapped on the conduit types, and certain solution schemes can be reused again and again. For example, when dealing with a complex multiplexing problem, the multiplexing could be divided into simpler multiplexing levels. The Accessors generating new keys to Sessions could the be relatively simple and alike.

The Conduits+ philosophy could be viewed as offering pipes, through which the communicating elements communicate. For example, in IPv6, there is a "pipe" in the Conduits+ graph between every two hosts communicating. If there is no communication between two hosts, the pipe between them does not exist. When communication between two hosts is requested, new objects, for example Sessions, are dynamically added in the graph. Therefore, the Conduits+ Framework can be seen as being memory efficient.

Regarding reusability, the Conduits+ framework could still be improved. The possibility of combining a collection of conduits into a bigger entity, that could be inserted elsewhere in a graph, would be of use. A main strength of the Conduits+ framework, however, is the idea that a protocol can be viewed as consisting of certain, well-defined elementary objects that capture the common characteristics of protocols. Providing such a framework enforces the protocol developers to think in a certain, perhaps more general, manner on protocol design.

The IPv6 protocol is stateless. This causes a slight semantic problem as some of the functionality is implemented with the Conduits+ States. Most of the functionality of a protocol is implemented in the States, so the framework seems to be more suitable to a protocol having many states.

Understanding object-oriented thinking is not easy. Also, learning a new framework is hard. This puts much responsibility on the design of a framework. The framework must be well documented and it should be carefully trained to the users. Otherwise, the benefits of the framework may not be understood, and the developers may use the framework in a inconsistent manner based on previous non-object-oriented design technique experiences.

6.4 Java experiences

Java offers useful features for programming in general and for protocol development in particular. The garbage collection facilities free the programmer from memory management concerns and lessen the amount of bugs in the code.

Java offers libraries with useful methods for telecommunication programming. The Java.net packet offers a class that represents for example Internet Addresses. The methods are convenient when manipulating IP Addresses. The packet also provides simple methods for using sockets.

Java offers portability. Every environment running the Java Virtual Machine can run Java code.

Some problems are related to Java and data communication programming. One problem is that there is no unsigned char, just byte arrays. As byte arrays are signed two's complement 8-bit integers, some kind of conversion must take place, as most values in data communication protocols are given as hexadecimal values. The bytes are always represented as 32 bit integers, which is a problem concerning computation.

Java offers a native environment for using native code, for example C code, to perform certain operations. The use of native code is however not standardized yet.

The JDK tool kit is not yet very mature. The library functions are not very well documented. The javac compiler is not very optimized and does not always even compile all the changed files. The error messages of the compiler are not very descriptive.

The threads of Java also cause problems. The scheduling policy is not precisely defined in Java. Therefore, one cannot rely on threads being pre-emptive.

Some concern can be arisen about the existence of Java decompilers. These decompilers decompile the Java byte code to java code. The two existing decompilers are named MOCHA and GUAVAC. The existence of decompilers can cause problem concerning copyright matters, and perhaps even concerning security matters. The problem could be somewhat mitigated by making the byte code more complex, for example by omitting method names in the byte code and providing numbers instead. This would make the decompiled code more unreadable.

Performance of the Java interpreter is not very good. However, using just-in-time compilers such as Kaffe seems to improve the performance much.

7 Conclusions

As computer network technology advanced and the use of distributed systems became more common, new security problems arouse. Internet Protocol Security (IPSEC) is a proposal for providing security services in Internet. IPSEC provides two security mechanisms, Authentication Header (AH) and Encapsulated Security Payload (ESP). AH is used to provide authentication, integrity and replay protection for IP datagrams, while ESP is used to provide authentication, integrity and confidentiality for IP datagrams.

Internet Protocol, version 6 (IPv6) is the new version of the Internet Protocol. It is based on the current version of IP, IPv4. IPv6 offers expanded addressing capabilities, header format simplification, improved support for extensions and options, flow labelling capabilities and network security services. These security services are being offered through IPSEC. The philosophy of IPv6 is that all ordinary packets should require as less processing as possible. All special handling of a packet, for example fragmentation, is done through extension and option headers.

Conduits+ is a framework for building telecommunication protocol software. Its primary goal of supporting design of reusable object-oriented software is offered by building the framework using Design Patterns.

The goal of this project was to implement a prototype of IPv6 and IPSEC that is portable, reusable and modular. An object-oriented approach was taken and the Java Language was chosen as the implementation language. The Conduits+ Framework was chosen as an experimental framework for the implementation.Main emphasis was put on the implementation architecture, which should be efficient and elegant. Performance was considered an issue only concerning the architecture, not the prototype. The IPv6 implementation would be minimal and key management was chosen to be manual.

The prototype was implemented in Sun Solaris 2.5. The specification work of IPSEC is in progress and the implementation is based on the Internet Drafts as they were in the beginning of the study. New drafts have been released and the prototype is therefore out of date. As part of the prototype, the Conduits+ framework was implemented in Java using the original Conduits+ ideas as the base and some new ideas by Pekka Nikander. An Ethernet adapter was implemented in C code, and the implementation was tested in a real network between two hosts.

Frameworks are a powerful tool when addressing reusability of code. By providing the infrastructure, the framework decreases the amount of standard software that must be developed. The developers can focus on the area where they can add most value to their code. Changes in the framework generally updates in every application.

Conduits+ is a framework for building telecommunication protocol software. It consists of four basic elements, that can be combined in various ways. The adapter is used to interface the conduits to the outer world. The Protocol is used to implement the state machine of a communication Protocol. The Mux is used to multiplex connections and the ConduitFactory is used to dynamically add new conduits to the structure. The data flowing through the conduits are called information chunks.

This thesis presents a Conduits+ architecture for the entire IPv6 and IPSEC. It seems to capture the dynamic and flexible nature of IPv6. Each IPv6 header is a module of its own. These modules all have the same structure. To support the arbitrary number of extension headers in IPv6, a loop is incorporated in the Conduits+ graph. The framework can be seen as memory efficient, as new objects, for example

Sessions, are dynamically added. The thesis suggest breaking up the protocol functionality in relatively small parts, thereby mapping the solutions on the simple conduit types. The idea of breaking a protocol into elementary objects that capture the common characteristics of protocols, is a main strength of the Conduits+ framework.

Java offers portability. Any machine implementing the Java Virtual Machine can run Java byte code. Java offers libraries with useful methods for protocol programming. The garbage collection facilities free the programmer from memory management concerns and lessens the amount of bugs in the code. The use of native code in Java is not standardized yet and the scheduling policy is not precisely defined. Concerning telecommunication programming, adding an unsigned char type would be useful. The Java Developers Kit is not very mature and needs improvement. Performance can be significantly improved by using just-in-time compilers such as Kaffe.

The experiences of using the Conduits+ framework and the Java Language are mainly good. The Conduits+ framework is still not entirely mature and can be evolved. The Java environments need improvement and optimization to get better performance. The main goals of the study - reusability, portability and modularity - seem to be possible to achieve using Conduits+ and Java.

8 References

[AAL96] Aalto, T., A Unix Streams Implementation of the Internet Protocol Security, master's thesis, 1996

[AN96] Aalto, T., Nikander, P., A Modular, STREAMS based IPSEC for Solaris 2.x Systems, Nordic Workshop on Secure Computer Systems, November 1996

[AG96] Arnold, K., Gosling, J., The Java Programming Language, Addison-Wesley Publishing Company, Inc., 1996

[AMO95] Amoroso, E., Fundamentals of Computer Security Technology, Prentice Hall, Inc, 1994

[BUD96] Budd, T., An Introduction to Object-Oriented Programming, Addison-Wesley Publishing Company, Inc., 1991

[CG96] Chang, S., Glenn, R., HMAC-SHA IP Authentication with Replay Prevention, Work in Progress, NIST, May 1996

[CHE94] Cheswick, W., Bellovin, S., Firewalls and Internet Security: Repelling the Wily Hacker, Addison-Wesley, Reading. Mass., 1994

[COM95] Comer, D. E., Internetworking with TCP/IP, Volume I, 3rd edition, Prentice Hall International, 1995

[COS91] Comer, D. E., Stevens, D., Internetworking with TCP/IP, Volume II, Prentice Hall International, 1991

[CP95] Cotter, S., with Potel, M., Inside Taligent Technology, Addison-Wesley Publishing Company, Inc., 1995

[EHJ96] Engel, R., Huni, H., Johnson, R., A Framework for Network Protocol Software, [FLA96] Flanagan, D., Java in a Nutshell, O'Reilly and Associates, Inc., 1996

[GHJV95] Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns, Addison-Wesley Publishing Company, Inc., 1995

[GM96] Gosling, J., McGilton, H., The Java Language: A White Paper, http://sunsite.nus.sg/hotjava/1.0alpha3/doc/overview/java/index.html

[HFFS95] Housley, R., Ford, W., Farrell, S., Solo, D., Internet Public key infrastructure, Work in progress, November 1995

[HUI96] Huitema, C., IPv6 The New Internet Protocol, Prentice Hall, New Jersey, 1996

[HUG96] Hughes, J., Combined DES-CBC, HMAC and Replay Prevention Security Transform, Work in Progress, April 1996.

[HÖG96] Höglund, K., A Conduits+ and Java Implementation of the Internet Key Management Protocol, Work in Progress, 1997

[I-D96a] Atkinson, R., Security Architecture for the Internet Protocol, Work in Progress, November 1996

[I-D96b] Atkinson, R., IP Authentication Header, Work in Progress, June 1996

[I-D96c] Chang, S., Glenn, R., HMAC-SHA IP Authentication with Replay Prevention, Work in Progress, November 1996

[I-D96d] Oehler, M., Glenn, R., HMAC-MD5 IP Authentication with Replay Prevention, Work in Progress, November 1996

[I-D96e] Atkinson, R., IP Encapsulating Security Payload (ESP), Work in Progress, June 1996

[I-D96f] Hughes, J., Editor, Combined DES-CBC, HMAC and Replay Prevention Security Transform, Work in Progress, September 1996

[I-D96g] Doraswamy, N., Editor, Combined 3DES-CBC, HMAC and Replay Prevention Security Transform, Work in Progress, November 1996

[I-D96h] Orman, H. K., The OAKLEY Key Determination Protocol, Work in Progress, May 1996

[I-D97] Maughan, D., Schertler, M., Schneider, M., Turner, J., Internet Security Association and Key Management Protocol (ISAKMP), Work in Progress, February 1997

[KAR91] Karila, A., Open Systems Security - an Architectural Framework, Dissertation, Helsinki University of Technology, Espoo, 1991

[KEN89] Kent, S., Comments on “Security Problems in the TCP/IP Protocol Suite”, ACM Computer Communication Review, Vol. 19, No. 3, July 1989, pp. 10-19

[KMS95a] Karn, P., Metzger, P., and Simpson, W., “The ESP DES-CBC Transform”, RFC 1829, Qualcomm, Inc., Piermont, Daydreamer, August 1995.

[KMS95b] Karn, P., Metzger, P., and Simpson, W.,”The ESP Triple DES Transform”, RFC 1851, Qualcomm, Inc., Piermont, Daydreamer, September 1995.

[KMS95c] Karn, P., Metzger, P., and W. Simpson, “The ESP DES-CBC plus MD5 Transform”, Work in Progress, Qualcomm, Inc., Piermont, Daydreamer, September 1995.

[KS96b] Karn, P., and Simpson, W., ICMP Security Failures Messages, Work in Progress, Qualcomm, Inc., Daydreamer, April 1996.

[MAR97] professor Martikainen, O., Tik-109.450 Object oriented protocol development, course lecture, Helsinki University of Technology, 1997

[MAS95] Maughan, D., Schertler, M., Schertler, M., Turner, J., Internet Security Association and Key Management Protocol (ISAKMP), NSA, Terisa Systems Inc., June 1996

[NBS77] National Bureau of Standards. Data encryption standard, January 1977, Federal Information Processing Standards Publication 46.

[NBS80] National Bureau of Standards. DES modes of operation, December 1980, Federal Information Processing Standards Publication 81.

[NPSH97] Nikander, P., Pärssinen, J., Sahlin, B., Höglund, K., A Java Based Framework for Cryptographic protocols, Work in Progress, June 1997

[ORM96] Orman, H. K., The Oakley Key Determination Protocol, Work in Progress, University of Arizona, May 1996.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, USC/Information Sciences Institute, September 1981

[RFC1883] Deering, S., Hinden, R., Internet Protocol, Version 6 (IPv6) Specification, RFC 1883, December 1996

[SCH96] Schneier, B., Applied Cryptography, Second Edition, John Wiley & Sons, New York, 1978.

[SS92] Sieworek, D., Swarz, R., Reliable Computer Systems, Design and Evaluation, Digital Press, 1992

[STA95] Stallings, W., Network and Internetwork Security Principles and Practice, Prentice Hall, New Jersey, 1995.

[STE94] Stevens, W. R., TCP/IP Illustrated: the protocols, Addison-Wesley Publishing Company, Inc., 1994

[TAL95] Taligent, Inc., The Power of Frameworks, for Windows and OS/2 developers, Addison-Wesley Publishing Company, Inc., 1995

[TRI] Trivedi, K. S., Probability and Statistics with Reliability, Queueing and Computer Science Applications, Prentice Hall, ISBN 0-13-711564-4