HELSINKI UNIVERSITY OF TECHNOLOGY

Department of Computer Science

Kaj Höglund

A Conduits+ and Java Implementation of the Internet Key Management Protocol

Master's thesis submitted for approval June 3, 1997 for the degree of Master of Science.

Supervisor Prof. Arto Karila

Instructor M.Sc. Pekka Nikander

Author and name of the thesis:

Kaj Höglund: A Conduits+ and Java implementation of the Internet Key Management Protocol
Date: 03.06.1997 Number of pages: 71

Faculty: Department of Computer Science
Professorship: Computer Networks (Tik-110)

Supervisor: Prof. Arto Karila

Instructor: M.Sc. Pekka Nikander

As more implementations for securing the communication over the Internet exist, the need for Key Management increases. A proper Key Management protocol is needed to negotiate authenticated and secure session keys for security protocols such as Internet Protocol Security (IPSEC).

Internet Security Association and Key Management Protocol (ISAKMP) is the key management protocol chosen for use in the Internet to provide services for both IPSEC and IPV6 as well as for other security protocols. ISAKMP provides for security concepts that aid in negotiating Security Associations (SA) and cryptographic keys in an Internet environment. Besides of creating SAs and cryptographic keys, ISAKMP provides procedures for authenticating the communicating peers as well as functionality for preventing security attacks.

The ISAKMP protocol as well as the Oakley key determination protocol which uses the ISAKMP framework to establish secure session keys are described in the paper. A description of how Key Management is connected to Internet security is also given. The creation of Security Associations which are agreements between communicating peers on how they protect the communication between them, is the main task of ISAKMP. Before creating Security Associations for other security protocols ISAKMP must create a Security Association for protecting it's own further traffic. After this, other SAs can be created. The establishment of an ISAKMP SA is called a phase1 negotiation, while a phase2 negotiation is used for establishment of other security protocol SAs.

The basis for this text is an ISAKMP prototype implementation that can establish Security Associations using the Oakley protocol's two modes: Main- and Quick mode. These two modes correspond with ISAKMP phase1 and phase2 negotiations. The objectives of the implementation is mostly portability which is why the implementation uses a Conduits+ framework as it's skeleton. The Conduits+ framework is a framework for building communication protocols and tries to factor out the common structure that exist in communication protocols in general. The framework is an object-oriented one and uses well known and good proven design patterns to provide for more reusability. Design patterns are ment to help software designers to make good design choices that are specific to the problem but also general enough to address future problems and requirements. As ISAKMP functions as a finite state machine, the Conduits+ framework suits well in this implementation. Most of the protocol specific implementation is done in the session and state objects of the ISAKMP protocol, so the framework becomes quite reusable.

As the Conduits+ framework is an object-oriented framework, an object-oriented programming language was needed. The Java programming language was chosen because of it's many positive features. Java e.g. provides libraries for easy network connectivity and is a reliable and robust language. A Java program can run on all existing platforms that implements the Java Virtual Machine which makes the program highly portable. Network protocols can also use the Java multithreading feature to serve many clients at one time.

Keywords: Internet security, ISAKMP, Oakley, Key Management, framework, Java, Conduits+

TEKNISKA HÖGSKOLAN

SAMMANDRAG AV DIPLOMARBETET

Författare och arbetets namn:

Höglund, Kaj: A Conduits+ and Java Implementation of the Internet Key Management Protocol

Datum: 03.06.1997

Antal sidor: 71

Avdelning:
Data avdelningen

Professur:
Datornät (Tik-110)

Övervakare: Professor Arto Karila

Handledare: DI. Pekka Nikander

En ökning av program som garanterar datasäkerheten i Internet leder till behov av program som kan generera äkta och hemliga sessionsnycklar för bruk av dessa säkerhetsprogram.

Internet Security Association and Key Management Protocol (ISAKMP) är ett nyckelkontrollerings-protokoll (Key Management) som valdes till att erbjuda service åt både IPSEC, IPV6, såväl som åt andra säkerhetsprotokoll. ISAKMP erbjuder säkerhetsmekanismer med vars hjälp man kan skapa säkerhetsassosiationer (Security Association) och kryptografiska nycklar för bruk i Internet miljön. Förutom dessa erbjuder ISAKMP skydd mot en del datasäkerhetsattacker som ofta används mot protokoll.

Detta arbete granskar ISAKMP protokollet samt Oakley protokollet som är ett protokoll för utbyte av kryptografiska nycklar mellan olika parter. Oakley använder sig av ISAKMP ramen (framework) för att skapa hemliga sessionsnyckar. Skapandet av säkerhetsassosiationer är en av ISAKMPs huvuduppgifter. En säkerhetsassosiation är en överenskommelse mellan olika parter om hur kommunikationen mellan dessa bör skyddas. Förrän ISAKMP kan skapa dessa assosiationer åt andra säkerhetsprotokoll måste ISAKMP skapa en egen assosiation som skyddar dess egna kommunikation.

Denna text baserar sig på en ISAKMP prototyp implementation med vars hjälp man kan skapa säkerhetsassosiationer genom användandet av Oakley protokollets Main- och Quick moder. Med hjälp av Main moden skapar man säkerhetsassosiationer för ISAKMP bruk medan Quick moden används för skapandet av assosiationer för andra säkerhetsprotokoll. Största målsättningen med detta arbete var att skapa en implementation som är lätt att överföra från en platform till en annan. Därför används en Conduits+ ram som en bas för implementationen. Denna ram är avsedd för bruk vid planering och implementering av datakommunikationsprotokoll. Strävan är att hitta en gemensam struktur från dessa protokoll som kan användas på nytt i olika implementationer. Conduits+ ramen baserar sig på ett objekt orienterat tankesätt och använder sig av kända Design Patterns med vars hjälp man strävar att göra implementationen allt mera användbar. Med hjälp av dessa Design Patterns strävar man att göra besult som är problemspecifika men ändå tillräckligt allmänna för att kunna användas i kommande problemlösningar. Eftersom ISAKMP fungerar som en finit tillståns-maskin passar Conduits+ ramen bra som en bas för ISAKMP implementationen. På detta sätt kan största delen av protokoll implementationen fungera utanför ramen vilket leder till en hög återanvändning.

Användandet av en objekt orienterad ram förutsätter användandet av ett objekt orienterat språk. Java valdes på grund av dess många goda egenskaper. Java erbjuder ett klassbibliotek med vars hjälp man kan tex. använda TCP/IP stacken. Java är robust och pålitlig samt oberoende av använd platform. Multithreading är en egenskap som tillåter klienter att kontakta protokollservern samtidigt.

Nyckelord: Datasäkerhet i Internet, ISAKMP, Oakley, Nyckelkontroll, Conduits+

Tekijä ja työn nimi:

Höglund, Kaj: A Conduits+ and Java Implementation of the Internet Key Management Protocol

Päivämäärä: 03.06.1997 Sivumäärä: 71

Osasto:
Tietotekniikan osasto

Professuuri:
Tietokoneverkot (Tik-110)

Työn valvoja: Prof. Arto Karila

Työn ohjaaja: DI. Pekka Nikander

Internetin tietoturvan varmistavien sovellusten lisääntyessä kasvaa myös tarve avaintenhallintasovelluksille. Kunnollista avaintenhallintasovellusta tarvitaan neuvottelemaan aitoja ja salaisia istuntoavaimia tietoturvaprotokollien, esimerkiksi Internet Protocol Security (IPSEC), käyttöön.

Internet Security Association and Key Management Protocol (ISAKMP) on avaintenhallintaprotokolla joka valittiin luomaan avaintenhallintapalveluja sekä IPSEC:n, IPV6:n sekä muiden tietoturvaprotokollien käyttöön Internetissä. ISAKMP tarjoaa turvamekanismit joiden avulla voidaan luoda tietoturva-assosiaatioita ja kryptograafisia avaimia Internet käyttöön. Näiden lisäksi ISAKMP tarjoaa vastapuolen todennuksen sekä pystyy torjumaan eräitä tietoturvahyökkäyksiä.

Tämä työ tarkastelee ISAKMP protokollaa sekä Oakley avaintenvaihtoprotokollaa joka käyttää ISAKMP viitekehystä luodessaan salaisia istuntoavaimia. Tietoturva-assosiaatioiden luonti on eräs ISAKMP:n päätehtävistä. Tietoturva-assosiaatio on eri osapuolten välinen sopimus siitä miten näiden välinen yhteys turvataan. Ennen kuin ISAKMP pystyy palvelemaan muita turvaprotokollia assosiaatioiden luonnissa täytyy ISAKMP:n luoda itselleen oma turva-assosiaatio jonka avulla turvataan ISAKMP:n oma liikenne.

Tämän tekstin perustana toimii ISAKMP prototyyppitoteutus jolla voidaan luoda tietoturva-assosiaatioita käyttäen Oakley protokollan Main- ja Quick moodeja. Main moodin avulla luodaan ISAKMP:n turva-assosiaatio kun taas Quick moodin avulla luodaan turva-assosiaatioita muille tietoturvaprotokollille. Suurin tavoite tässä työssä painottuu toteutuksen siirrettävyydelle jonka vuoksi työn runkona käytetään Conduits+ viitekehystä. Tämä viitekehys on tarkoitettu tietoliikennesovelluskäyttöön jonka avulla pyritään eristämään näistä sovelluksista löytyvä yhteinen rakenne jota voidaan käyttää uudelleen monissa eri toteutuksissa. Viitekehys perustuu olioperustaiseen ajattelutapaan ja käyttää hyviksi todettuja hahmoja (Design Pattern) joiden avulla pyritään lisäämään sovelluksen uudelleenkäytettävyyttä. Hahmojen avulla pyritään tekemään suunnitteluratkaisuja jotka ovat ongelmakohtaisia mutta tarpeeksi yleisiä jotta niitä voidaan käyttää tulevissa ongelmissa ja vaatimuksissa. Koska ISAKMP toimii äärellisenä tila-automaattina, Conduits+ viitekehys sopii hyvin ISAKMP:n rungoksi. Näin suurin osa itse protokollan toteutuksesta tapahtuu viitekehyksen ulkopuolella jolloin uudelleenkäytettävyysaste on korkea.

Koska käytettävä viitekehys on olio-perustainen, tarvitaan sovelluksen luomiseen olio-perustainen ohjelmointikieli. Java kieli valittiin sen monen hyvän piirteensä vuoksi. Java tarjoaa luokkakirjastoja joiden avulla voi helposti käyttää TCP/IP pinoa hyväkseen. Java on luotettava ja robusti ohjelmointikieli joka on käyttöjärjestelmäriippumaton. Monisäikeisyyden avulla on helppo tehdä palvelinohjelmia jota palvelevat monia asiakkaita yhtäaikaisesti.

Avainsanat: Internetin tietoturva, ISAKMP, Oakley, Avaintenhallinta, viitekehys, Conduits+

Acknowledgements

This work was done to investigate the possibilities to use a communication protocol framework as an aid in implementing cryptographic protocols. Though just scratching the surface, the work has introduced many interesting aspects of computer system security. Therefore I wish to thank professor Arto Karila for giving me the opportunity to get acquainted with this topic. I also wish to thank my instructor Pekka Nikander for his help during the work. Especially his help in implementing and understanding the Conduits+ framework has been significant. Special thanks also to my work colleagues Timo and Bengt for keeping up right spirit.

Table of Contents

List of Figures

Terms and Abbreviations

AH	Authentication Header
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
CBC	Cipher Block Chaining mode
Conduits+	Framework for communication protocol software
cryptographic transformation	Cryptographic algorithm and set of rules surrounding the use of cryptographic algorithm in a particular context
DES	Data Encryption Standard. A symmetric key encryption and decryption algorithm
DNS	Domain Name System. The on-line distributed database system used to map human-readable machine names into IP addresses and vice versa
ESP	Encapsulating Security Payload
Ethernet	A local area network technology
FDDI	Fiber Distributed Data Interface. A token ring network based on fiber optics
FTP	File Transfer Protocol
HMAC	Hash function based Message Authentication Code
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
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
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
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

Network ID	Network Identifier
Oakley	The Oakley Key Determination Protocol
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
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
socket	The abstraction provided by UNIX operating system that allows an application program to access the TCP/IP protocols
SPI	Security Parameter Index. An unstructured opaque index which is used in conjunction with the IP destination address to identify a particular security association
Symmetric encryption	A form of cryptosystem in which encryption and decryption are performed using the same key
TCP	Transmission Control Protocol
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
WWW	World Wide Web. A global hypertext based information system

1 Introduction

Internetworking means the technology used to connect separate physical networks enabling communication between them. The need for internetworking is growing all the time as users and organizations have greater need to share data. This can be the need to use multimedia services on demand to homes or to establish a video conference between remote sites. The communication is more and more breaking out from the intranet to the Internet.

The technology used in the Internet is based on the TCP/IP protocol suite. It has evolved from a research funded by the Defense Advanced Research Project Agency (DARPA) in the middle of 1970. The TCP/IP protocol suite allows computers from different vendors to communicate with each other, and provides means of internetworking and routing traffic between different networks. The suite consists of various protocols that provide services for applications using the Internet. Such services are e.g. electronic mail, file transfer, remote login, and the World Wide Web (WWW).

As TCP/IP is network independent it can be used in future communication. TCP/IP can run on top of ethernet, ATM or FDDI for example. As the Internet is more and more used for distribution of secret and confidential information, a means for protecting this data must be implemented. There are already a couple of implementations that perform this task. These security services can be implemented at various positions in the protocol stack e.g. at the application layer or at the network layer.

This text is based on the providence of ISAKMP key management services for the Internet Protocol security (IPSEC), which is an Internet Engineering Task Force (IETF) standard for network layer security of the Internet. Internet Security Association and Key Management

Protocol (ISAKMP) defines a set of procedures for authenticating the communicating parties, creating and managing security associations, key generation techniques, and threat mitigation (e.g. denial of service and replay attacks). All of these are necessary to establish and maintain secure communications in the Internet environment. [ISAKMP97].

Oakley Key Determination Protocol is a protocol used by two communicating parties to agree on secure and secret keying material. The basic key exchange mechanism is based on the Diffie-Hellman key exchange algorithm, as described in chapter 2.9, combined with RSA based authentication. Oakley is designed to be a compatible component of ISAKMP. This means that Oakley data is carried in ISAKMP payloads.

ISAKMP and Oakley papers have not yet evolved into RFCs. These drafts are valid for a period of maximum six months, and they can be updated or replaced at any time. The implementation, described in chapter 5, is based on the drafts dated November 22, 1996 and May, 1996, while this text is based on an ISAKMP draft published February 21, 1997 and on the draft: "The Resolution of ISAKMP with Oakley" [ISAOAK97] published in February 1997.

The goal of this work was to implement two exchange types of the Oakley protocol, namely Main mode and Quick mode. Main mode corresponds directly to ISAKMP's Identity protecting exchange, while Quick mode uses ISAKMP’s Security Association to protect itself.

1.1 Implementation Description

The implementation uses an object oriented Conduits+ protocol development framework which aids in writing reusable protocol software. The framework tries to factor out the common structure that exist in communication protocols in general, and make it possible to concentrate on the protocol specific parts. The framework consists of two types of objects, namely conduits and information chunks. The conduits are software building blocks that eventually make up a conduit graph representing the set of the protocols implemented. The information chunks are data or communication messages that the protocols get as input or produce for output.

The framework uses well known design patterns to make the implementation even more reusable. The patterns are meant to help software designers to make good design choices that are specific to the problem at hand but also general enough to address future problems and requirements.

The protocols were implemented with the Java programming language, developed by Sun, to test Java's suitability for protocol development. Java is an object-oriented programming language designed for use in the Internet. Java is machine independent as it is compiled into byte-code which can be run on any system that implements the Java Virtual Machine.

The problem at hand is thus to try to gain understanding and knowledge about key management protocols designed for Internet use and problems that might incur from their implementation. As the drafts describing these protocols are updated quite often, the implementation of a framework makes the impact of these changes less critical. The experiences gained from work like this might help to construct products for the Internet that are both secure and easy to use.

1.2 Outline of this Paper

The rest of this thesis is organized as follows: Key management is discussed in general in chapter 2, and the Isakmp and Oakley protocols in chapter 3. Chapter 4 gives a presentation of the architectural elements including the Conduits+ framework, design patterns, and a general description of the Java programming language. Chapter 5 presents the implementation of the protocols. The class model, architecture, and design choices are presented. Chapter 6 is used for the results achieved with this work and the experiences gained from it.

2 Internet Security and Key Management

This chapter presents the key management problem for cryptographic protocols. It deals with Internet security and key management in general, their purpose, and what they try to accomplish. The chapter should function as a short introduction into Internet security and key management before discussing the specific protocols used in this implementation in the next chapter. Most of the text in this chapter originates from [SCH96], which can be recommended to anybody interested in applied cryptography.

Section 2.1 discusses what security services are available today, how they are chosen and used. Chapter 2.2 then highlights some aspects of data encryption while chapter 2.3 discusses aspects key management has to deal with. Chapter 2.4 provides for a short description of what security attacks can be used against key management.

2.1 Available Security Services

Security services are different functionality by which one can protect data in various ways. One person might be interested in pertaining anonymity, while others want the secrecy of data. One goal of key management is to negotiate the parameters being used between entities to achieve these services. Key management can also in itself provide some of the below mentioned services. The definitions below are from [ATK96a] and [STA95].

Authentication: A security property that ensures that the origin of received data is, in fact, the claimed sender. In case of a single message, the assurance that the message came from the source it claims to be from. In the case of a longer session between two entities one first has to ensure at start-up that the other party is the one she claims to be, and during the session the assurance that somebody has not performed a masquerade attack by pretending to be another user.

Authentication mechanisms fall into two categories of strength - weak and strong. Passwords are an example of a weak authentication mechanism. Digital signatures using DSA or RSA are public key based strong authentication mechanisms. ISAKMP requires the usage of strong authentication.

An authenticated key exchange protocol provides strong forward secrecy if disclosure of long term secret keying material does not compromise the secrecy of the exchanged keys from previous communications. This means that compromise of a single key will permit access to only data protected by that key. For Perfect Forward Secrecy (PFS) to exist the key used to protect transmission of data must not be used to derive any additional keys, and if the key used to protect transmission of data was derived from some other keying material, that material must not be used to derive any more keys [ISAOAK97].

Integrity: A property that ensures data is transmitted from source to destination without undetected alteration. If the order of transmitted data also is ensured, the service is termed connection-oriented integrity. Integrity can be applied to a single message, a stream of messages or to a selected part of a message.

Confidentiality: A security property that ensures that communicated data is not disclosed to other, unauthorized parties. Traffic flow confidentiality tries to hide source- and destination addresses, traffic frequency, and other traffic characteristics.

Non-repudiation: A security property that ensures that a participant in a communication or computation cannot later deny having participated in the communication.

Access control: The ability to limit and control the access to host systems, applications and data. Uses authentication to verify the entity trying to gain access.

Availability: Requirement that computer services are available to authorized users when needed.

Security Association: A simplex (one-way) logical connection, created for security purposes. All traffic traversing an SA is subjected to the same security processing at the transmitter and receiver.

2.1.1 Security Associations

A Security Association (SA) is in principle a list of management parameters used by communicating entities to secure the communication. An SA is one-way, meaning that two communicating peers have two SAs in use, one for each direction. The task of key management is to establish and update the variables in an SA, while the task of a security protocol is to use these variables for its purposes. The sending entity finds the appropriate SA by using its user identifier and the destination address, while the receiving part uses the Security Parameter Index (SPI) and the destination address to find the correct SA.

Example parameters in an SA could be: SPI, destination address, authentication algorithm with mode and key, encryption algorithm with mode and key, lifetime of the association and some sensitivity label.

2.1.2 Security Policy

The use of different algorithms and security services is dictated by a local security policy, which is a set of requirements that must be met by any secure system implementation. In the case of key management a security policy affects the creation of Security Associations (SA) between hosts. An SA between two or more entities is identified by a Security Parameter Index (SPI). The combination of the SPI value and a destination address uniquely identifies an SA.

An initiator who wants to establish an SA needs to consult the local security policy to find out the set of acceptable values for variables being negotiated. Such variables include encryption algorithm, hash algorithm, signature algorithm, key establishment algorithm as well as user- and host oriented certificates. The responder checks the incoming messages, and consults its local security policy for an appropriate answer. The initiator can provide many alternatives it supports and the responder then chooses the services that suits its own policy.

2.2 Different Encryption Methods

The encryption and decryption of data is achieved through a mathematical algorithm. The algorithm uses a key and the original message to produce a new encrypted message. In the same way, the original message is restored from the encrypted message using the key. There are two types of encryption algorithms: symmetric- and public key algorithms. These differ from one another in the way they use keys. In a symmetric algorithm the encryption and decryption key can be deduced from one another. Often the key is the same for both encryption and decryption. This means that only the communicating peers must know the key used.

In public key- or asymmetric algorithms the encryption key differs from the decryption key. As the name implies, the encryption key can be made public, while the other key must be kept secret. Messages encrypted with the public key can only be decrypted by a party who has the private key. Public key algorithms can also be used for digital signatures. In this scheme a party who wants to sign a message can decrypt it with her private key. Another person who can open this encryption with the public key of the sender can be pretty confident of the message's origin. Public-key-, or asymmetric algorithms also increase security, because the private key does not need to be transported anywhere.

2.2.1 Key Lengths

When generating keys there is always the discussion of what is the minimal key length required. Often the encryption target specifies how strong the encryption must be. If transferring money over the network, the encryption must hold for the duration of the transfer but, e.g. the secrecy of personal medical records should hold for the lifetime of the individual. If keys, strong by today's standards, are used with medical records, they may easily be broken after 10 years, as technology advances. There are other ways to break an algorithm, but one should strive to use such an algorithm that is well known and good proven by the experts of cryptography. An attack against such an algorithm must be done by using brute force. This means that every possible key combination must be gone through until the right one is found. "The sizes of encryption keys are measured in bits and the difficulty of trying all possible keys grows exponentially with the number of bits used. Adding one bit to the key doubles the number of possible keys" [BLA96]. The size of the key thus places an upper bound on the strength of the cryptosystem.

As already mentioned, both symmetric and asymmetric cryptosystems exists. Asymmetric systems can be used for encrypting the encryption key used in symmetric systems. Symmetric cryptosystems are used more widely for encryption because they are much faster than the asymmetric ones. "Public-key algorithms are slow. Symmetric algorithms are generally at least 1000 times faster than public-key algorithms" [SCH96].

Hardware that can be used for key testing gets cheaper and cheaper all the time. The more money one can invest in equipment, the faster one can run through the key-space and test all possible combinations. More sensitive data that interests corporations with lots of spendable money requires longer keys. The same applies to security that must hold for a long period of time.

Many applications use too short keys. 40 bits is of no challenge for today's computers. And 'Moore's Law' says that computing power doubles every 18 months for a given cost. [BLA96] predicts from this rule that if data should still be secure after 20 years, the key lengths should be about 14 bits longer than those that would resist an attack today. They recommend using a minimum key-length of 90 bits for symmetric cryptosystems.

2.3 Key Management and Distribution

This chapter discusses things key management has to consider when generating, authenticating and transferring keys.

2.3.1 Key Generation

It is important to use a strong mechanism when generating keys because the security of the cryptosystem rests in the key. If generating keys e.g. for an algorithm that uses 56-bit keys the generation mechanism should be such that any 56 bit string could be the key. Thus there would be 2^56 possible choices.

Poor key choices are vulnerable to a dictionary attack where the attacker has a dictionary of common keys. Good keys are random bit strings generated by some automatic process. Every possible key must be equally likely. The key should be generated by a reliably random source or by a cryptographically secure pseudo-random-bit generator. For more information on these generators, see [SCH96].

Some encryption algorithms have weak keys. These are keys that are less secure than other keys, usually only a couple of keys in a huge key-space. Despite of this fact, it is worth testing for the existence of these keys and perform a regeneration of the key, if a weak one is found [SCH96].

2.3.2 Certificates

Key management needs to deal with issues such as how to know that a public-key really belongs to a claimed user. The solution to this is public key certificates which bind each key to a specific user. The problem with certificates is the need for an authority that creates and distributes these certificates.

A certificate contains information such as a version number and a serial number. An algorithm identifier tells which algorithm is used to sign the certificate. Issuer is the certification authority that created and signed the certificate. A certificate has a period of validity after which it expires. A subject field identifies the user to whom this certificate refers, and the public key information of the subject tells the actual public key. A field containing the signature that verifies the correctness of the certificate is also needed. The signature is a signed hash over the other fields in the certificate. The hash can be produced by using a hash algorithm which takes a variable length input and produces an output of a specified length. Even by knowing the hash and the hash algorithm, one cannot reproduce the input to the algorithm.

Security extensions to the Domain Name System (DNS) can provide resource records which are defined to associate public-keys with DNS names. In this way the DNS can function as a key distribution center. Authentication of the public-key is achieved by using digital signatures.

2.3.3 Key Transfer

When using a symmetric algorithm for encryption, all parties involved in the process need the same key. There must be some way of transmitting the key to all parties securely. There are various approaches in solving this problem. The X9.17 standard specifies two types of keys: key encryption keys that encrypts keys for distribution, and data encryption keys that encrypts messages between hosts. The key encryption keys must be distributed manually or they can be stored on tamperproof smart cards. Another way to distribute a key is that of splitting the key into different parts and sending those parts over separate channels to the recipient.

The use of a Key Distribution Center (KDC) can be adequate when using symmetric cryptography. The KDC generates and distributes session keys to parties requesting them.

In public key cryptosystems the key exchange is simpler. One party just encrypts the session key with the recipients public key and sends it over the network. Only the recipient can decrypt the key using her secret key. This scheme is vulnerable to a man-in-the-middle attack discussed in chapter 2.4

2.3.4 Lifetime of Keys

[SCH96] presents several arguments why an encryption key should be used for only a definite period. Reasons presented are:

- The longer a key is used, the greater the chance that it will be compromised.
- The longer a key is used, the greater the loss if the key is compromised.
- The longer a key is used, the greater the temptation for someone to spend the effort necessary to break it.
- It is generally easier to do cryptanalysis with more ciphertext encrypted with the same key.

Usually it is the security policy that determines the lifetimes of keys. When using fast communication links [SCH96] recommends updating the session keys at least once a day. The key encryption keys can be updated less frequently. Private keys used for digital signatures and proofs of identity must usually remain secret for a long time, even for many years. In these cases the old key must be stored somewhere when a new one is generated. This because someone might want to verify a signature on an old document.

2.3.5 Key Usage, Updates and Storage

When using software encryption, the key can get compromised if the computer stops the encryption task, writes all data to a disk, and starts operating on another task. All the time the operating system performs another task, the key is stored on the disk and can be compromised. One way to avoid this, is to assign such a high priority to the encryption process that no interrupts will happen. "The safest place in a software implementation is to store the key in kernel level data structures owned by the control device driver" [TAA96]. Hardware implementations are safer than software implementations. These devices know when a key has been tampered with and can be designed to erase that key.

When communicating sensitive material and large quantities, it is necessary to update the encryption key. This can be done by creating a new key altogether, or by creating a new key from the old one. If the updating process is frequent, the latter is more preferable as it saves distribution traffic. A one-way hash function can be used to produce a new key from the old one. All parties use the same hash function that gets as it's input the current key. The output, or part of it, can then be used as a new key. This does not work if the current key has been compromised, as the intruder can use the same hash function to produce the new key.

Safest key storage is achieved by using a tamperproof chip or a smart card. The card is inserted e.g. into a slot in the keyboard when the encryption key is needed. This eliminates the need to remember keys, or to store them in unsafe places. One should try to achieve that keys do not appear unencrypted outside the encryption device.

2.3.6 Key Management Protocols

The main goal of Key Management protocols is to produce authentic and secret keys. The keys should be produced in a way that old revealed keys does not reveal new ones and new revealed keys does not reveal old ones. These two properties are called Perfect Forward Secrecy (PFS) and Back Traffic Protection (BTP).

There are a number of good proven and heavily tested Key Management protocols which can be used to perform this task. One basic rule is that it is not worth trying to design a Key Management protocol oneself. This section presents two Key Management protocols, both of which are vulnerable to security attacks. The goal of presenting such protocols is to show the difficulty of designing secure Key Management protocols. Section 2.4 presents the attacks that can be used against these protocols.

The first one presented is called the Needham-Schroeder protocol which tries to establish a secret session key between two parties. The outline of the protocol is as follows:

A --> T: A, B, Na

T --> A: Ea(Na, B, Kses, Eb(Kses, A))

A --> B: Eb(Kses, A)

B --> A: Ekses(Nb)

A --> B: Ekses(Nb + 1)

The protocol needs three different parties: Alice (A), Trent (T), and Bob (B). Alice wants to establish a secret key to be used in communication between herself and Bob. She sends her name, Bob's name and a random nonce Na to Trent, a trusted third party who can generate good session keys. Trent generates Kses, the secret session key, and sends it encrypted back to Alice. In the same message he sends the same key encrypted with Bob's secret key. Alice can open the encryption done with her secret key and sends the other encrypted part to Bob. Bob decrypts the message and finds Kses, the session key. He also finds Alice's name, and thus knows from whom the message originated. Bob computes a random nonce value Nb, encrypts it with Kses, and sends it to Alice. Alice can open the encryption. She adds one to the nonce value, encrypts the message, and sends it back to Bob. This nonce generation prevents the replay-attack to take place where messages are recorded and re-sent at a later time.

Another example is the Diffie-Hellman key exchange protocol. This is a protocol where the keying material can be exchanged over an insecure channel. The peers agree on a large prime number n, and a generator g. The other party generates a random large integer x, and the other generates a random large integer y. Both x and y are secret values. The initiator computes $X = g^x \bmod(n)$ and sends X to the responder. The responder computes $Y = g^y \bmod(n)$ and sends Y to the initiator. Now the initiator can compute the key $k = Y^x \bmod(n)$ and the responder can compute $k = X^y \bmod (n)$. Both parties thus in fact compute $k = g^{x*y} \bmod (n)$ which becomes the session key.

2.4 Attacks Against Key Management Protocols

The availability of strong proven cryptographic algorithms have shifted the security attacks against key management. As encryption strives to provide secrecy, confidentiality, integrity, and availability attacks against security try to break them by using interruption, interception, modification or masquerading.

A Denial Of Service (DOS) attack tries to prevent access to system resources. Interruption is a variant of a DOS attack and tries to stop a service from being available to the user whereas by interception the attacker tries, by passively listening to network traffic, to gain services not meant for the attackers use. If the attacker, while listening to the network traffic, also modifies it, she performs a modification attack. Masquerading, as the name implies, is fooling another participant to believe she is communicating with a different person.

Access to system resources can be granted or denied by a reference monitor. This monitor gets as its input a request from a subject and produces as its output a grant- or deny answer. Figure 2.1 shows the basic principle of the reference monitor.

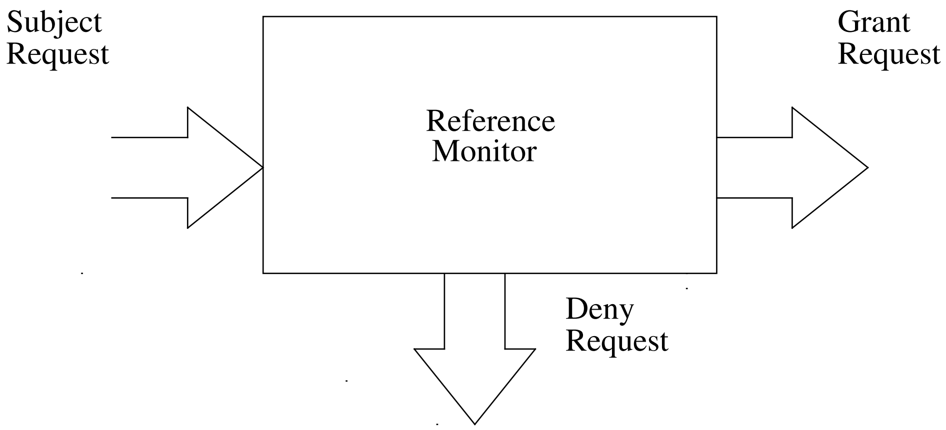

FIGURE 1. Reference monitor concept [AMORO94]

When an authorized user is not provided the service requested, a denial of service has occurred. V. Gligor says in [GLI83] that time should be included in the concept of denial of service. Every service should have a maximum waiting time under which a grant or deny answer should reach the subject. Clearly if that time elapses, a DOS attack might have occurred.

DOS attacks are the most difficult to address. The attack could be done e.g. by writing a virus program that when run reproduces itself and runs itself. By making the virus spread to other machines, makes this an effective way of increasing the load and eventually causing services to be interrupted. Another way is to send many large files as e-mail and bringing down the mail server. There are also weaknesses in the TCP/IP protocol suite that make it vulnerable to DOS attacks [BEL89]. By using SYN flooding one can block a TCP port and thus make it inaccessible to service requests. Also by sending a reset packet to a TCP port during the session, it can be closed.

Denial of service attacks against key management protocols strive to flood the protocol with requests from bogus source addresses. This attempt tries to make the service unavailable by consuming CPU resources.

By coming in the middle of two communicating peers, one can perform a Man-in-the-Middle attack. In this position one can modify, delete, or create new messages altogether. A man-in-the-middle can, of course, also perform a denial of service attack. Often this kind of attacks are performed by misleading other parties into believing that they are communicating with each other.

The Needham-Schroeder protocol is still vulnerable to a replay-attack despite the nonce exchange described in section 2.3.6. The attack can be achieved if an attacker can record the third message Eb(K,A) and somehow knows the old session key K. The attacker can now pretend to be Alice by replaying the old message. To prevent this attack, timestamps that prove freshness, can be used.

The strength of the Diffie-Hellman protocol, mentioned in the last section, is the difficulty of computing discrete logarithms, that is, even if one knows X, g, and n, it is difficult to compute x. Someone can still perform a man-in-the-middle attack and fool both the initiator and responder into believing that they communicate with each other. All the attacker has to do is: receive X from the initiator and compute a value Z of his own that is sent to the initiator. In the same way, he sends the value Z to the responder who sends him the computed value Y. Now the attacker can compute two different keys used in communication between both the initiator and responder. Man-in-the-middle can be thwarted by using digital signatures that verify the ownership of the key.

3 Internet Key Management Protocols

The Internet Security Association and Key Management protocol (ISAKMP) provides a general framework that can be used for negotiating Security Associations (SA) between hosts. As mentioned in section 2.1.1, an SA is a relationship between two or more entities where they agree on certain security services used for protecting the communication between these entities. ISAKMP, by itself, does not establish all the needed parameters, e.g the session keys. Instead, ISAKMP can be used in conjunction with various key establishment protocols. One such protocol is the Oakley key determination protocol. Oakley uses Diffie-Hellman key exchange method to create session keys on hosts. Resolution of ISAKMP with Oakley [ISAOAK97], describes how to use the ISAKMP framework with Oakley.

3.1 The ISAKMP Protocol

The ISAKMP protocol defines procedures and packet formats to establish, negotiate, modify and delete Security Associations. SAs contain all the information required for execution of various network security services, such as the IP layer services, transport or application layer services, or self protection of negotiation traffic. ISAKMP defines payloads for exchanging key generation and authentication data. These formats provide a consistent framework for transferring key and authentication data. The framework is independent of key generation techniques, encryption algorithms and authentication mechanisms. There may be different key exchange protocols, each with different security properties. However, a common framework is required for agreeing on the format of SA attributes, and for negotiating, modifying, and deleting SAs. ISAKMP serves as this common framework.

ISAKMP is intended to support the negotiation of SAs for security protocols at all layers of the network stack (e.g. IPSEC). By centralizing the management of the security associations, ISAKMP reduces the amount of duplicated functionality within each security protocol. ISAKMP can also reduce connection setup time, by negotiating a whole stack of services at once [ISAKMP97]. The placement of ISAKMP in a network architecture is shown in Figure 2 below.

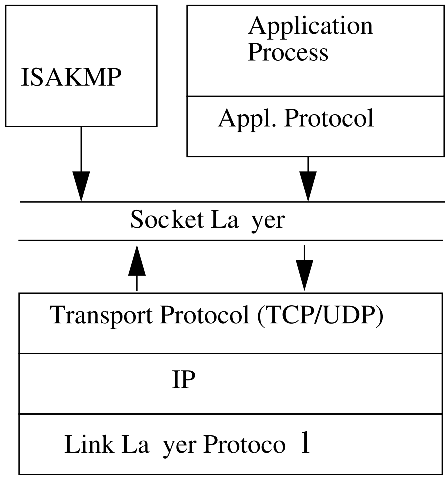

FIGURE 2. ISAKMP placement

3.2 Parameters Negotiated Between Entities

Basically, what is negotiated between entities are security associations. These associations must support different encryption algorithms, authentication mechanisms, and key establishment algorithms for all kinds of security protocols, e.g. IP security. Security associations must also support host-oriented certificates for lower layer protocols and user-oriented certificates for higher level protocols. Algorithm and mechanism independence is required in applications such as e-mail, remote login, and file transfer, as well as in session oriented protocols, routing protocols, and link layer protocols. ISAKMP provides a common security association and key establishment protocol for a wide range of security protocols, applications, security requirements, and network environments [ISAKMP97].

ISAKMP is not bound to any specific cryptographic algorithm, key generation technique, or security mechanism. This flexibility is beneficial for a number of reasons. First, it supports the dynamic communications environment described above. Second, the independence from specific security mechanisms and algorithms provides a forward migration path to better mechanisms and algorithms. When improved security mechanisms are developed or new attacks against current encryption algorithms, authentication mechanism, and key exchanges are discovered, ISAKMP will allow the updating of the algorithms and mechanisms without having to develop a completely new Key Management protocol [ISAKMP97].

3.2.1 IPSEC Parameters

Each IPSEC Security Association is uniquely identified by the combination of a Security Paramenter Index (SPI) and the destination address. As ISAKMP uses two phases for negotiation, it thus creates two Security Associations. One for protection of its own traffic, and the other for a security protocol (e.g. IPSEC). The SA attributes for IP security are defined in [ATK96a] and include the parameters listed below, but might include additional parameters as well:

- Authentication algorithm and algorithm mode being used with the IP Authentication Header (AH).
- Key(s) used with the authentication algorithm in use with the Authentication Header
- Encryption algorithm, algorithm mode, and transform being used with the IP Encapsulating Security Payload (ESP).

- Key(s) used with the encryption algorithm in use with the Encapsulating Security Payload
- Presence/absence and size of a cryptographic synchronization or initialization vector field for the encryption algorithm.
- Authentication algorithm and mode used with the ESP transform.
- Authentication key(s) used with the authentication algorithm that is part of the ESP transform.
- Lifetime of keys or time when a key change should occur.
- Source Address(es) of the Security Association.
- Sensitivity level (e.g. Secret, or Unclassified) of the protected data.

3.3 Services Provided by ISAKMP

ISAKMP supports key establishment and key generation by using public key cryptography. The Oakley key determination protocol is used in conjunction with ISAKMP to provide this service. Other protocols than Oakley can also be used, but Oakley is classified as mandatory in the ISAKMP draft. In Oakley the well-known Diffie-Hellman key exchange protocol is used for key generation. The benefit of this algorithm is that the key used for encrypting messages is based on information generated by both parties. The independence of keys from one key exchange to another provides for perfect forward secrecy.

ISAKMP provides authentication of the key exchange. This can be done during the protocol or after protocol completion. Authentication of the key exchange during the protocol is provided when each party provides proof it has the secret key before the end of the protocol. Proof can be provided by encrypting known data in the secret session key during the protocol exchange. Authentication after the protocol run must occur in subsequent communications. Authentication during the protocol is preferred so that subsequent communications are not initiated if the secret session key is not established with the desired party.

ISAKMP also provides back traffic protection which is concerned with the protection of previous traffic, even if keys used for protecting future traffic are compromised. This is done by using public key cryptography for the establishment of session keys.

ISAKMP supports symmetry in key exchange. An exchange is symmetric if either party can initiate the exchange and exchanged messages can cross in transit without affecting the key that is generated. This is desirable so that computation of the keys does not require either party to know who initiated the exchange.

3.3.1 Protection Services Offered by ISAKMP

ISAKMP defines a number of basic requirements for its authentication and key exchange components. These requirements guard against denial of service, reply, man-in-the-middle, and connection hijacking attacks. This is important because these are the types of attacks that are targeted against security protocols. Complete Security Association support, which provides mechanisms and algorithm independence, and protection from protocol threats are the strengths of ISAKMP [ISAKMP97].

Denial of service. ISAKMP uses so called anti clogging tokens or cookies to prevent DOS attacks from happening. Both the initiator and responder compute a cookie value of their own, which is then used throughout the SA establishment. The cookies are placed in the ISAKMP header and checked at reception of every message. This allows for detecting and discarding bogus messages without consuming excessive resources. Cookies are made up of a hash of IP addresses, UDP ports, current time and date. This should guard against an attacker using random IP source addresses [ISAKMP97].

Connection hijacking is prevented by linking the authentication, key exchange and security association exchanges. This linking prevents an attacker from allowing the authentication to complete and then jumping in and impersonating one entity to the other during the key and security association exchanges[ISAKMP97].

A **Man-in-the-Middle** can perform interception, insertion, deletion, and modification of messages. The linking of ISAKMP messages prevents the insertion attack. The ISAKMP protocol state machine is defined in such a way that deleted messages will not cause partial SA establishment. It clears all states and returns to idle if something erroneous is received. Reflection of old messages is also prevented by requiring a new cookie with time dependent material. SA creation with the intended party is ensured with strong authentication.

3.4 Payloads

ISAKMP payloads are the building blocks for ISAKMP messages. The exchange type dictates the ordering of the payloads in messages. Exchange types are described in more detail in section 3.5 that discusses the creation of a Security Association. This section highlights the more important payloads, the ones that are needed to establish a Security Association.

ISAKMP header: The header is placed in the beginning of each message. It is not in itself a payload but contains information required by the protocol to maintain state, process payloads and possibly prevent denial of service or replay attacks. Figure 3 shows the header format.

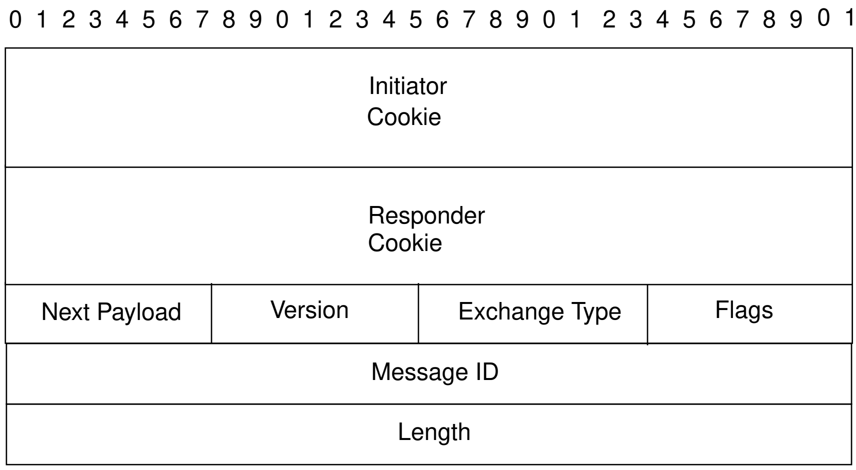

FIGURE 3. ISAKMP header format

The initiator and responder cookies remain fixed throughout an SA establishment. These are used to prevent denial-of-service attacks as described earlier. The next payload field tells which payload to expect next in the message, while the version tells which ISAKMP version this message was created by. A bit is set in the flags field when the payloads are encrypted. In such a case, the only unencrypted part of the message is the header. The message id is used to identify protocol state during a phase 2 negotiation. During phase 1 this value must be set to 0. The last field in the header indicates the length of the total message in octets.

Generic header provides for the ability to chain payloads together and contains three fields that identify the next payload, the length of the payload and a reserved field for future use.

SA payload: This payload is used to negotiate security attributes used between entities. The initiator can suggest security services according to a local security policy and the responder can pick from that list the most preferred ones. The SA payload defines a domain and situation under which the negotiation takes place. Proposal and transform payloads, which are in a way part of the SA payload, define the parameters to be created. The Domain of Interpretation (DOI) specifies the domain under which the negotiation is taking place. The most usual domain is Internet. The situation field is used to make policy decisions regarding the security attributes being negotiated.

Proposal payload defines the protocol for which the SA is negotiated, and informs how many transform payloads are chained together. Each proposal payload contains a security parameter index (SPI). The proposal payload is used by the peers to present to each other the security protocols and security mechanisms for use with the security association being negotiated. If the SA establishment negotiation is for a combined protection suite consisting of multiple protocols, then there must be multiple proposal payloads each with the same proposal number. The use of same proposal number provides an AND operation, meaning protocol1 AND protocol2, e.g. IPSEC AH and IPSEC ESP. If the proposal numbers are different, this provides for an OR operation, e.g IPSEC AH or IPSEC ESP. The protocol identifier field specifies which security protocol the SA is created for. Such security protocol could be e.g. ISAKMP, IPSEC AH or IPSEC ESP. The Security Parameter Index (SPI) uniquely identifies the SA [ISAKMP97]. Figure 4 shows the format of the proposal payload.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Next Payload	RESERVED	Payload Length	
Proposal #	Protocol - Id	SPI Size	# of Transforms
SPI (variable)			

FIGURE 4. ISAKMP proposal paylod format

Transform payload: The transform payload is one that carries information about the wanted parameters, or attributes in a Security Association. These attributes are domain of interpretation specific. It is the SA attributes field in the payload that carries information about these attribute values. Figure 5 shows the format of the transform payload.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Next Payload	RESERVED	Payload Length
Transform #	Transform - Id	RESERVED 2
SA Attributes		

FIGURE 5. ISAKMP transform paylod format

Key exchange payload: This payload supports a variety of different key exchange techniques. ISAKMP must support the Oakley protocol. Data required for session key generation is carried in the key exchange data payload.

Nonce payload: The nonce payload is used to guarantee the liveness during an exchange and protect against replay attacks, that is, recording data and replaying it at a later time. The nonce in itself contains random data generated by the transmitting entity.

Identification payload: This payload contains Domain of Interpretation specific data needed to exchange identification information. The ID Type field specifies which type of identification is used. This can be e.g. an IPv4 address. The data would then be the IP address of the current host.

Signature payload: This payload carries data signed by the digital signature function selected in the Security Association payload. This provides a means for ensuring the identity of the communicating peer.

Hash payload: The hash payload is used in the phase 2 of the negotiation in establishing Security Associations for other security protocols. The payload carries data generated by the hash algorithm selected in phase 1. This data can be used for verifying message integrity or for authentication.

3.5 SA Establishment

As already mentioned, a Security Association is an agreement between entities on how to protect the communication between them. ISAKMP supports two different kinds of SA creations. First, it needs to create an SA for itself, i.e. between the communicating ISAKMP entities. This is needed to be able to protect ISAKMP's own further traffic. When this SA is established, another security protocol can ask ISAKMP to establish an SA for its needs. ISAKMP then uses its own security mechanisms to negotiate security parameters for the other protocol. The initiating entity can provide multiple proposals for the parameters it supports. This is dictated by the local security policy. It is up to the receiver of such a message to pick the parameters that are supported by its local security policy. An example of a SA establishment can be found in chapter 5.1 in figure 5.

3.6 The Oakley Protocol

Oakley defines a method to establish an authenticated key exchange. The definition describes how payloads are constructed, the information they carry, the order in which they are processed, and how they are used. Oakley is a key exchange protocol whereas ISAKMP provides a framework for authentication and key exchange but does not define them. ISAKMP is designed to support many different key exchanges, Oakley being the only mandatory one. This section is based on [ISAOAK97], which describes how to use those protocols in conjunction.

Oakley defines a series of key exchanges it calls modes which provide certain security services such as perfect forward secrecy, identity protection and authentication. The main goal of Oakley is to work in conjunction with ISAKMP to provide authenticated keying material for both ISAKMP and other security associations such as IPSEC AH and ESP. Oakley defines three different modes of communication: Main-, Aggressive-, and Quick mode. This study is limited to concern the Main- and Quick modes.

The relationship between ISAKMP's negotiation phases and Oakley's modes is straightforward. ISAKMP's phase 1 is where the two ISAKMP peers establish a secure and authenticated channel with which to communicate. The Oakley Main mode accomplishes such a phase 1 negotiation. Oakley quick mode is used when a phase 1 negotiation is already done. This is when a Security Association is negotiated for another security protocol such as IPSEC, or when generating fresh keying material. A single phase 1 negotiation can be used to protect more than one phase 2 negotiations.

Oakley defines a set of attributes that are negotiated as a part of an ISAKMP Security Association. Such attributes are encryption algorithm, hash algorithm, authentication method and information about the group over which to do Diffie-Hellman.

The Oakley exchanges adapt to ISAKMP payload formats. The Oakley Main mode is an instantiation of the ISAKMP identity protect exchange. To indicate this, the exchange type field in the ISAKMP header is set to a value indicating identity protection. The Oakley Quick mode exchange has no direct analog in ISKAMP.

Implementations can use three different authentication methods in Oakley Main mode. The authentication can be based on digital signatures, public-key encryption or pre-shared keys. This chapter describes Oakley Main mode based on digital signatures. For more information on the others, see[ISAOAK97] and [OAKLEY].

The Oakley Main mode exchange consist of three round trips. The first round trip negotiates the security policy to be used and consists of HDR and SA payloads. The second round trip exchanges the Diffie-Hellman public values and consists of HDR, KE, and N payloads. The last roundtrip then completes the exchange by authenticating the Diffie-Hellman exchange. This roundtrip uses HDR, ID, and SIG payloads.

The second roundtrip produces keying material (SKEYID) from which different keys can be obtained. For example the keying material used by an implementation that bases authentication on digital signatures is obtained by computing: SKEYID = prf(Ni | Nr , g^{xy}) where prf is the pseudo random function defined in the ISAKMP SA payload, or if the prf is optionally left unnegotiated, the HMAC version of the negotiated hash algorithm must be used. The result of the Main mode exchanges is the following keying material:

SKEYID_d = prf(SKEYID, g^{xy} | CKY-I | CKY-R, 0)

SKEYID_a = prf(SKEYID, SKEYID_d | g^{xy} | CKY-I | CKY-R, 1)

SKEYID_e = prf(SKEYID, SKEYID_a | g^{xy} | CKY-I | CKY-R, 2)

CKY-I and CKY-R are the cookie values transported in the ISAKMP header. To authenticate the key exchange the initiator computes HASH_I and the responder computes HASH_R in the following fashion:

$$HASH\_I = prf(SKEYID, g^{xi} \mid g^{xr} \mid CKY\text{-}I \mid CKY\text{-}R \mid SAp \mid IDii)$$

$$HASH\_R = prf(SKEYID, g^{xr} \mid g^{xi} \mid CKY\text{-}R \mid CKY\text{-}I \mid SAp \mid IDir)$$

When the authentication is done by using digital signatures, both hash values are signed and verified.

A phase2 negotiation used for SA creation for other security protocols is done by using Oakley Quick mode. This is not a complete exchange in itself, and is done under the protection of the ISAKMP SA negotiated in phase1. Quick mode is essentially an exchange of nonces that provide replay protection. The nonces are used to generate fresh key material and to prevent replay attacks from generating bogus SAs. The Quick mode consists of one and a half round trips. The first round carries HDR, HASH, SA, Ni and optionally KE payloads. Everything, except the HDR, is encrypted. The half roundtrip from initiator to responder consists of a HDR and an encrypted HASH payload. If perfect forward secrecy is not required, the KE payload can be left out from the first roundtrip. The values carried in the three HASH payloads are computed in the following fashion:

$$HASH(1) = prf(SKEYID\_a, M\text{-}ID \mid SA \mid Ni\ [\ \mid KE\ ])$$

$$HASH(2) = prf(SKEYID\_a, M\text{-}ID \mid SA \mid Nr\ [\ \mid KE\ ])$$

$$HASH(3) = prf(SKEYID\_a, 0 \mid M\text{-}ID \mid Ni \mid Nr)$$

If the KE payload is not used, the new keying material is obtainable from: $KEYMAT = prf(SKEYID\_d, SPI \mid Ni \mid Nr)$. Otherwise it is obtainable from: $KEYMAT = prf(SKEYID\_e, g(gm)^{xy} \mid SPI \mid Ni \mid Nr)$.

4 Architectural Elements

Architectural elements are the elements that help build the implementation. This chapter describes the three main components used in the ISAKMP implementation. The sections are written to include examples that help understanding the implementation description in the next chapter. Section 4.1 describes design patterns and is based on [GAMMA], which contains a description of a collection of well known design choices. The section also presents a few patterns that can be used in the ISAKMP protocol architecture.

While section 4.2 describes how various design patterns can be used in conjunction with the Conduits+ framework, section 4.1 tries to give an overview of the pattern concept. Section 4.3 then describes the Java programming language.

4.1 Design Patterns

According to Gamma et al. [GAMMA] : *Design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.* The patterns are meant to help designers to make good design choices that are specific to the problem at hand but also general enough to address future problems and requirements. By using these design patterns the architecture can be made smaller, simpler, and more understandable. [GAMMA] discusses the pattern solutions in terms of objects and interfaces, and gives each pattern four essential elements:

- A **pattern name** is a handle that can be used to describe the design problem, its solutions, and consequences in a word or two. Naming the pattern increases the design vocabulary. This helps in communicating and talking about these choices.
- A **problem** statement describes what one is trying to solve and the solution to that problem. It might give a description of

a poor design choice and give a list of conditions that must be met before it makes sense to use the pattern.

- A **solution** provides the building blocks that makes up a good design in this particular problem. It tells how to combine these blocks together and describes their responsibilities without going into details of a specific solution.
- The **consequences** describe what is gained and what is lost by applying a particular pattern. These might include trade-offs e.g. in time and space.

As an example of the above mentioned scheme could be mentioned e.g. the Factory pattern. The pattern name is Factory and it defines an interface for creating an object, but lets subclasses decide which class to instantiate. Often used in frameworks which are responsible for creating new objects.

The problem usually is that an application specific subclass needs to be instantiated and it is hard to predict which subclass should be instantiated. The problem is that a framework only knows about abstract classes, which it cannot instantiate.

The solution is to move the knowledge of the instantiation out of the framework and create a Factory method responsible for instantiation. The Factory method then returns an instance of the wanted class.

The consequences of this pattern is that it provides hooks for subclasses. Creating objects inside a class with a factory method is more flexible than creating an object directly. The Factory pattern can be used together with the Prototype pattern which clones the needed subclass. [GAMMA]

4.1.1 Pattern Classification

The patterns can be classified to families of related patterns. This classification is done using two criteria: **purpose** and **scope**. The purpose tells what the pattern does and the scope specifies whether the pattern applies to classes or to objects.

Class patterns deal with the classes and their relationships and thus rely on inheritance while object patterns rely on object relationships. Class patterns are more static in nature, i.e. they are fixed at compile-time while object patterns can be changed at run-time, i.e. are more dynamic in nature.

The purpose divides the patterns into **creational-, structural-,** or **behavioral** patterns. Creational patterns, as the name implies, are concerned with object creation while structural patterns deal with the composition of classes or objects. Behavioral patterns in turn describe how objects and classes interact with each other.

4.1.2 Some More Common Patterns

The remainder of this section introduces some patterns that can be used in the ISAKMP protocol implementation. The text contains some Conduits+ framework definitions explained more thorougly in section 4.2.

The Conduit itself (see section 4.2.1) with its Protocol subclass (see section 4.2.1.3) is an example of a Composite pattern, which belongs to the family of structural patterns. The pattern is used to represent part-whole hierarchies. As the conduit graph can be quite complex with composite protocol classes or individual conduit objects this pattern lets clients treat them all uniformly. Clients to the Conduit class should not have to make the distinction between individual or composite objects. Benefits to this scheme is that it makes the client simple and makes it easier to add new kinds of components . It can also make the design overly general.

The Mux Conduit (see section 4.2.1.1) can be reused in different places of the conduit graph by using the so called strategy pattern. The intent of the pattern is to define a family of algorithms , encapsulate each one , and make them interchangeable . This lets the algorithm vary independently from the clients that use it. This is done in the ISAKMP implementation by giving each Mux an Accessor that is responsible for returning a key that the Mux needs for multiplexing . In this way the algorithm is separated from the object, which can be reused. This pattern can successfully be used when many related classes differ only in their behavior, or the client needs different variants of an algorithm. Also when an algorithm uses data that clients should not know about, is implementable with this pattern.

The intent of the State pattern is to allow an object to change behavior when it changes state . This is a common situation in communication software that are implemented as finite state machines . One such example is the ISAKMP protocol. At each state the protocol should perform different tasks to incoming and outgoing data and must thus change behavior at run-time . The solution is to let the current state object take care of the data manipulation. The state object then takes care of state transition to the next state .

The State pattern is often used in conjunction with the Singleton pattern because there is always only one instance of a state class at any time . The Singleton pattern is used to make sure that only one instance of a state class exists . It creates a new instance of the state only if none has yet been created.

The Adapter element (see section 4.2.1.3) in the Conduits+ framework uses the Adapter pattern. The intent of the Adapter pattern is to provide an interface for messages coming from outside the framework. This provides for interaction between classes with incompatible interfaces. ISAKMP must be used on top of the UDP protocol but using the Adapter pattern it can as easily be run on e.g. TCP. The request clients call can be the same, only the Adapter provides a different interface. The picture below shows the function of an Adapter relying on object composition.

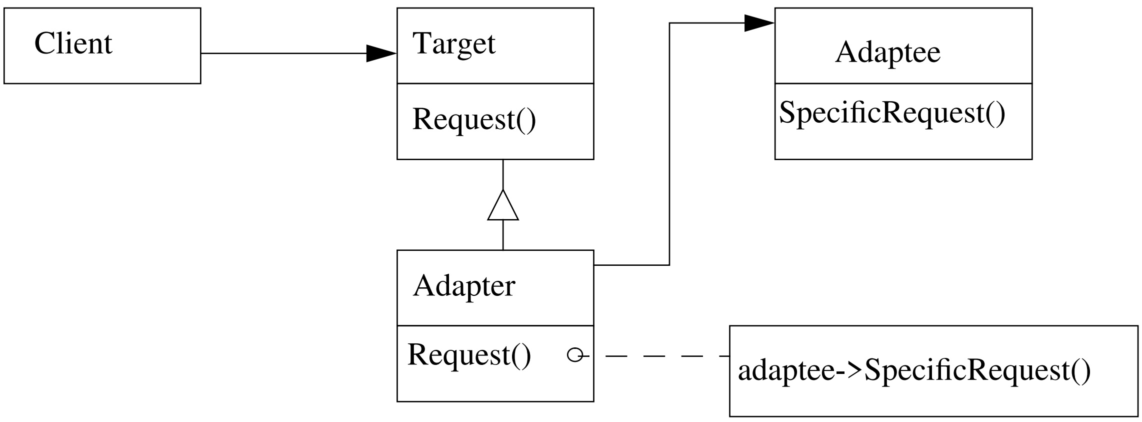

FIGURE 6. Adapter functionality

The data that flows through a Conduit graph can be represented using the Visitor pattern. Visitors can carry data or make changes to the conduits graph. The Visitor object contains operations to be performed on different conduits elements. The Visitor knows from its current position in the graph which operation to perform. The elements just need to accept a Visitor and tell where it has arrived.

The creation of a new Conduit element is done by using the Factory pattern. The ConduitFactory is responsible for creating a Visitor that will traverse the graph making changes to it. The factory uses the Prototype pattern for element creation. The intent of the Prototype pattern is to create new objects by copying existing prototypes. When a new prototype is needed the client asks the prototype to clone itself and to return that copy to the client.

4.2 Conduits+

In this section we describe a framework which can be used when creating communication protocols. The framework is based on the general work of Hüni, Johnson and Engel [COND+]. The framework provides a means for building protocol software. It to factors out the common structure that exists in these protocols, and make it possible to concentrate on the protocol specific parts. The result should be a common building block that can be reused in different implementations. To achieve reusability the framework relies on already existing design patterns as described in section 4.1.

Hüni, Johnson and Engel claim two benefits from using design patterns:

- The protocol specific software parts become simpler and easier to maintain and evolve since the framework provides the infrastructure operations.
- The result is a framework for network protocol software, in other words, a generalized software architecture for communication protocols. Documenting this framework helps communicate the underlying model and how it works.

Our framework is called Conduits+, and it tries to be a black-box framework. A black-box framework is a framework which is composing instances of classes to achieve component reusability. A black-box framework achieves its functionality by assembling or composing objects to get more complex functionality. Object composition requires that the objects being composed have well defined interfaces. The term black-box comes from the fact that no internal details of objects are visible. Object composition is defined dynamically at run-time through objects acquiring references to other objects [GAMMA].

Conduits+ is not all the way black-box, but relies also on white-box techniques, relying partly on inheritance to achieve reusability.

The framework consists of two types of objects: conduits and information chunks. The conduits are software building blocks that eventually make up a conduit graph representing the protocol implemented. The information chunks are data the protocol gets as input or produces for output. The graph can be reconstructed dynamically at run time as more conduits elements are needed.

4.2.1 Conduit Types

There are five different types of conduits, all of which are common building blocks in protocol software. The five types are: Adapter, Multiplexor, Session, Protocol, and ConduitFactory. All of these elements have two sides: sideA and sideB. The conduits can be connected, by attaching the sides to other conduits, which then become their neighboring conduits. An Adapter has only its sideA connected, while a Mux can have several conduits connected to its sideB. A Session and a ConduitFactory both have exactly one sideA and one sideB connected.

The data traverses the conduit graph from one component to another by using their sides. It can enter a conduit from either sideA or sideB and typically leaves the conduit at the opposite side it came from.

4.2.1.1 Mux

The **Mux** is a conduit that can have several conduit elements connected to its sideB. When the Mux receives data from its sideA conduit, it knows based on some multiplexing criteria which sideB conduit to forward the data to. Typically the sideB conduit is a Session, but it can also e.g. be another Mux. Data arriving from any of the sideB conduit is passed along to sideA. It should be easy to install and remove sideB conduits as they can be created and destroyed at run time.

4.2.1.2 Session

The **Session** is the hart of the implementation and is responsible for the producing, consuming and testing of data. The Session is often implemented as a finite state machine, and thus has to remember its current state. As already mentioned the Session has exactly one neighboring conduit connected on each of its sides.

The problem is how to install new conduits to, or uninstall old conduits from sideB of a Mux, and what to do with data that's destination address refers to a Session that has not yet been created. The framework solves this problem by attaching a **ConduitFactory** on sideB of each Mux. Each Mux has exactly one ConduitFactory, and zero or more other sideB conduits. If data arriving from sideA of a Mux has no destination sideB conduit to be forwarded to, the data is sent to the ConduitFactory. The factory then creates a new instance of the needed Session and sends the data back to the Mux. Now the Mux is able to route the data to its destination.

4.2.1.3 Adapter

An **Adapter** is needed to connect the framework with the world "outside". This is why the Adapter only uses its sideA to attach itself to the conduit graph. SideB functions as an interface to some other software or hardware implementation. For example, ISAKMP runs on top of the UDP protocol and uses an Adapter to connect to the socket interface for communication with the network.

4.2.1.4 Protocol

A Protocol acts as a container encapsulating several interconnected conduits . The Protocol can be used when changes constituting of several Conduits elements need to be done on the Conduit graph. The Protocol then contains these elements already "glued" together and can be inserted into the graph by attaching it between some already existing Conduits elements .

4.2.2 An Example

The following graph shows all the elements described above . This is a typical ISAKMP conduit graph in the beginning of a protocol execution when only one session element has been installed on sideB of two multiplexors .

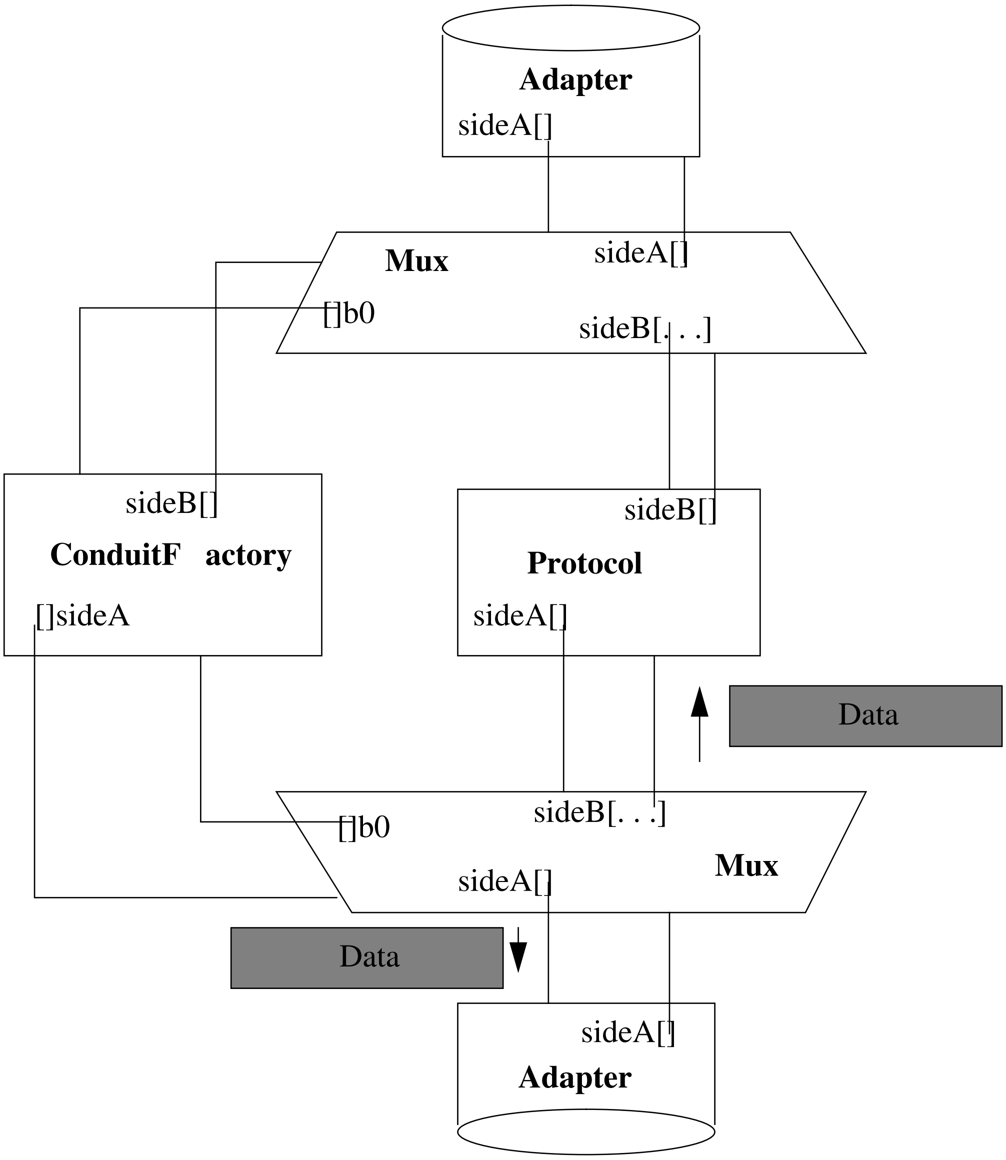

FIGURE 7. Typical Conduit graph

4.2.3 Usage of Design P atterns

The reusability can still be improved by using existing design pat- terns . These patterns e .g. make it possible to use the same Mux c lass everywhere in the implementation. T o be able to do this , the frame- work needs to separate the functionality that is specific to a certain

Mux from the Mux class. This can be done using the Strategy pattern which separates the algorithm from the object that uses the algorithm. This is done by giving each Mux an Accessor that is responsible for accessing the key needed for proper multiplexing of incoming data. Thus, the Accessor extracts the key from the data, and the Mux sends the data to the sideB conduit that has the matching key. In this way the Mux class no longer has to be subclassed into different types of muxes.

By using the State pattern one can represent each state the Session has by a separate object. The pattern specifies that the Session in itself does not alter or modify data. Instead, the state objects handle these tasks. The Session can be made as simple as providing only one method, one for accepting incoming data. The Session does not touch the data, instead the framework defines a MessageTransporter class which has the responsibility to carry data around the conduit graph. When the MessageTransporter arrives to the Session, the Session just calls the transporters appropriate method to inform the transporter where it has arrived, and where it came from. The transporter then calls the current states apply method. This method does the needed thing with the data the transporter carries. When done, the state object updates the Session objects state object to the next one.

The state objects should be sharable between many Session objects which means that variables like counters, timers etc. need to be stored in the Session object and not in the state objects. These variables are then used and manipulated from the state objects. By using the Singleton pattern the framework ensures that there is only one instance created of a state class. Thus, they will not have to be created or destroyed dynamically during run time.

The Messagetransporter is not the only object that is needed in traversing the graph. The framework also needs an Installer that is responsible for making changes to the graph. The solution is to create an abstract Visitor class and derive two subclasses MessageTransporter and Installer from it. The Installer class is the one responsible for the changes to the graph. When a Mux does not have the appropriate conduit element attached on sideB, it sends the MessageTransporter to its ConduitFactory. The factory uses the Prototype pattern to create a copy from a prototypical conduit. The responsibility of the Installer is to attach the new prototype to sideB of the appropriate Muxes. The above implementation is achieved by using the Visitor pattern.

4.3 The Java Programming Language

Java is a new programming language designed for use in the Internet and with the World Wide Web. As the Conduits+ framework was chosen to be one part of the implementation, an object-oriented language was needed. Java has many features that made it an attractive choice for this project. This section tries to highlight some Java features that are important when creating communication software. Most points are from [JAVA] which discusses the Java programming language.

Java is an object-oriented language and suits fine with the Conduits+ framework. It is a technique which focuses on the data, its objects, and the interfaces to these objects. A software object has a state and a behavior. The state is maintained in variables and the behavior is implemented through its methods. The methods defined in an object provides for the interface to other objects. Other objects use these interfaces when sending messages to the object in question.

Java is a distributed language which provides libraries e.g. for easy network connectivity. The libraries provide easy access to both TCP and UDP protocols through connection oriented- and connectionless socket interfaces. Java applications can open and access remote objects across the network as easily as local ones. This is clearly an essential part of communication software where many entities must be in contact with each other.

Communication software should also tolerate fault and error situations. An attacker could try to cause errors to happen on purpose but the implementation should be robust enough to tolerate these attacks. Java tries to be a reliable and robust language. It checks for problems in the code at an early stage, and later at run-time, in order to eliminate situations that can cause errors. The linker performs many of the same checks that the compiler did to guard against mismatch problems. One step in the way to achieve robustness is to forget all about pointers. This eliminates the possibility to overwrite memory and corrupt data.

The Java Developers Kit (JDK 1.1) contains security related enhancements. This security API supports e.g. digital signatures, hash algorithms, key management and access control lists. All these services are needed in the ISAKMP implementation, which made Java an attractive choice for this project.

Object-oriented programming tries to produce reusable software. The program should be able to execute on many different platforms with small modifications. Even better if the program could do this with no modifications at all. The same version of a Java application can run on all existing platforms, i.e. Java is machine independent. The Java code is compiled into byte-code format which can run on any system that implements the Java Virtual Machine. The Java Virtual Machine is a kind of software implementation of a Central Processing Unit (CPU) designed to run Java byte-code.

When the performance of interpreted byte-code is not enough, the byte-code can be translated into machine code for a particular CPU.

This enhances the speed of the program to nearly as fast as native C and C++.

Multithreading is a feature incorporated into Java that allows multiple parallel processes. Java provides a Thread class that supports methods to start, run, stop, and check the status of threads. By synchronizing execution the programmer can ensure that different threads do not update variables in an unwanted fashion. This is a wanted feature in communication software where many clients can contact one server.

5 Implementation

This chapter describes the implementation part of the Master's Thesis. First, a description of Security Association creation is presented. A description of interfaces towards the Security Policy Daemon (SPD) is described as well as the functionality of the ISAKMP state machine. Section 5.2 presents the implementation of the ISAKMP Conduits+ framework and section 5.3 presents the class model of the implementation describing what classes are needed to achieve the functionality. Section 5.4 then presents in more detail how the implementation works. The goal is to give the reader an understanding at a lower level about how messages traverse the Conduit graph and what functionality is needed in various Conduits element.

5.1 SA Establishment

The ISAKMP protocol is implemented as a socket server using the Java DatagramSocket class for sending and receiving messages between two communicating peers. The server runs on top of the UDP protocol and awaits for incoming messages at port number 5003.

A server can either be the initiating part or the responding part. If a server is the initiating part it needs to get the initial start-up command from another program, typically a Security Policy Daemon (SPD). The SPD module is a program that takes care of local security policy. The SPD knows the peer entity it wants to contact and also its own security policy. The SPD then informs ISAKMP of the destination address and the parameters to negotiate.

The SPD Adapter at the ISAKMP side supports a method for invoking initial ISAKMP SA creation, and also methods for invoking SA creation for other security protocols. Thus it is the task of the SPD implementation to know what Security Associations already exist.

Each SA is identified by a Security Parameter Index. The SPD must check the SA existence for a certain destination address, and call the appropriate ISAKMP method. If there is no SA created, an initial ISAKMP SA creation must be performed, but if one already exists, SPD can directly call the method that creates an SA for AH or ESP or some other security protocol.

When ISAKMP creates Security Associations between entities, it returns the identifying SPI back to the SPD. This SPI value and the destination address uniquely identify the SA. The SPD can then use this SPI value to check for existing SAs. Every transform(AH or ESP) needs its own SA. It is ISAKMP's task to negotiate these SAs for the SPD.

SAs for AH or ESP cannot be negotiated unless an ISAKMP SA exists. It is thus up to the SPD to call for the ISAKMP SA negotiation.

By using the SPI value as a search index in a hash table, ISAKMP can find the ISAKMP session that created the initial ISAKMP SA between entities. The session knows all the security parameters and keys to be used for protecting a new SA creation

The ISAKMP server is implemented as a finite state machine, see figures 8 and 9. Depending on if the server is the initiating part or the responding part, the server can be in different states. Both initiator and responder start from state0. In state0 the initiator creates the initial message which includes a proposal for security parameters for protection of further communication. The server then sends the message to its peer, and changes its own state to state1.

At the receiving side, the message appears at the UDP Adapter which is an interface towards the connectionless socket above UDP. The Adapter receives the message and, is responsible for its decoding. The message is transported through the conduit, and a new IsakmpSession is created for it. This session is initially in state0.

By using the State pattern described earlier, state0 is responsible for checking the correctness of the incoming packet. State0 also creates a response message. When this message is sent to the initiator, the responder changes to state2.

The initiator, or more correctly, its state1 object can now perform a check on the incoming message to see which security services the responder accepted. If an agreement on the parameters was achieved, the initiator creates the message for roundtrip 2 which exchanges the Diffie-Hellman public parameters ($X = g^x \bmod(n)$), and moves on to state3.

The State2 object at the receiving end can now compute the secret key used in the communication. The receiver sends a similar message with its public Diffie-Hellman parameter to the initiator. The initiator's state3 object can now compute the secret key, which should be equal to the one computed by the responder. Actually the key computed, functions as a key-space from which different keys can be computed. Such keys are: encryption-, hash-, and signature keys. The hash- and signature keys are used in the last roundtrip to sign a hash over the message. This is done to authenticate both peers.

When all three roundtrips are performed, the ISAKMP session at both ends remembers the parameters negotiated and sends the SPD implementation the SPI value for this communication. Figure 8 shows how the ISAKMP SA is created, how the states change between roundtrips as well as which payloads are transferred. Refer to section 3.4 for a description of the payloads .

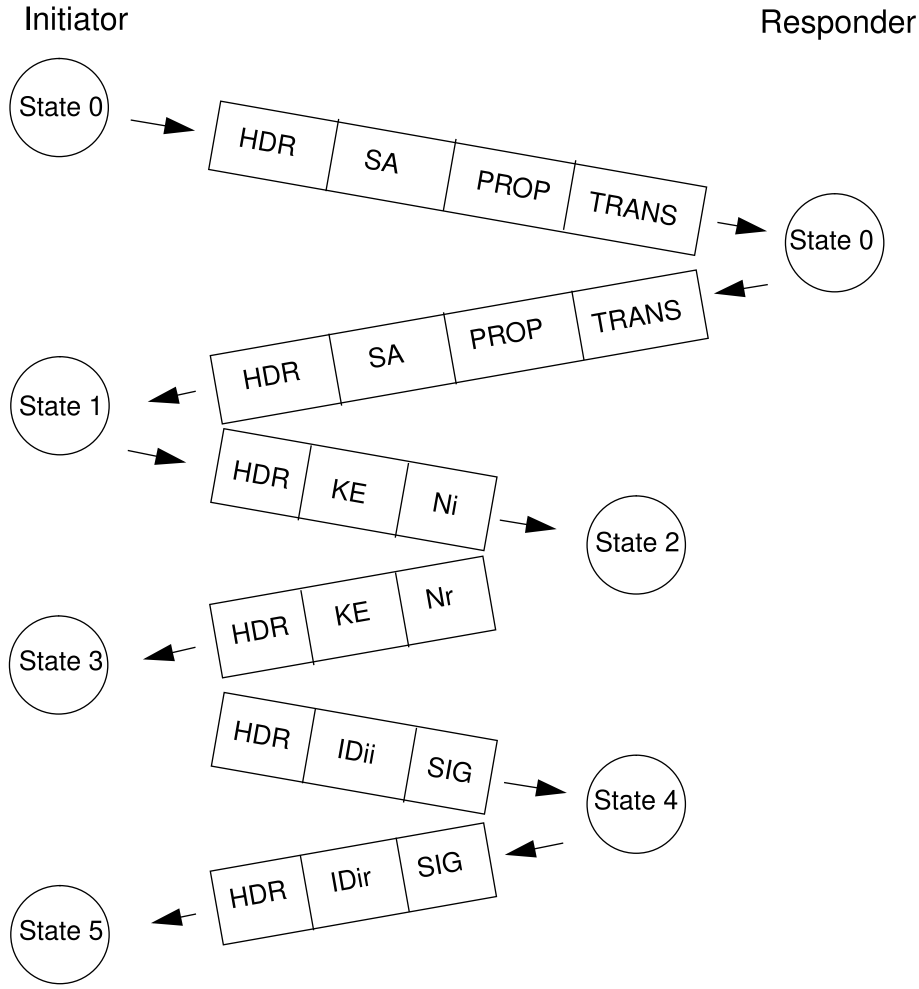

FIGURE 8. State transition and packets in a phase1 negotiation

The Oakley Quick mode is used in conjunction with ISAKMP phase2 negotiation to create an SA for other security protocols . If an SPI does not exist, SPD must initiate the creation of the ISAKMP SA. However , if the SPI exists , the SPD can directly call a method that creates an SA for either AH or ESP . The session that created the ISAKMP SA can be found from a hashtable and the keys and algorithms stored in this session are used for securing the communication.

One and a half roundtrips are needed to complete the phase2 negotiation. The initiator and responder create AH or ESP sessions depending on the request from SPD . As can be seen from figure 9, the states follow the same sc heme as in a phase1 negotiation. The initiator sends the first message encrypted to the responder . As the responder has the same encryption key as the initiator , it can decrypt the message . After c hec king and accepting the parameters in the SA pa yload, the responder replies with a similar message . The parameters negotiated in phase2 are passed to SPD , whic h uses these to secure the communication between the entities .

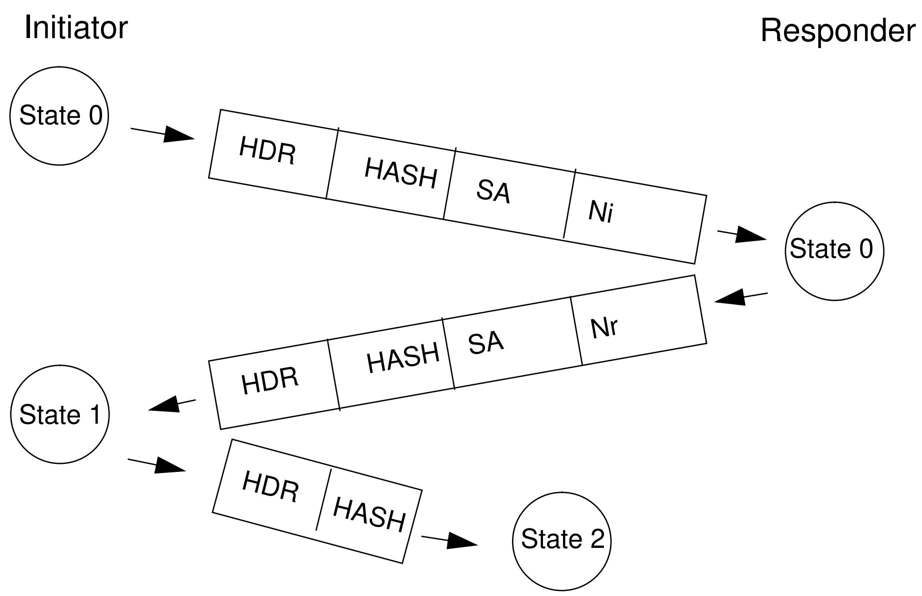

FIGURE 9. State transition and packets in phase2 negotiation

5.2 ISAKMP Conduits+ Framework

In this section we describe what parts make up the ISAKMP Conduits+ framework and describe their functionality. We concentrate more on the Conduit objects and their responsibilities. In section 5.4 we describe in more detail how the methods are implemented. At start-up, a basic static framework is built. These building blocks are the necessary parts needed to get things started.

Adapters at both ends of the framework guarantee that input can be received to the framework. The SPDAdapter works as an interface towards the SPDmodule. It receives requests to create Security Associations between entities. At the other end resides an UdpAdapter that receives messages from a communicating peer. The UDP-Adapter's sideA is at start-up attached to a Cookie Mux's sideA. Similarly, the sideA of the SpdAdapter is attached to sideA of another Cookie Mux. These Muxes have the function to route incoming requests to right sessions. They are aided in their task by so Accessors which extract out a key, the cookie, from incoming messages. Every Mux has its own Accessor, which provides for this functionality. The Mux uses the key as an index to a hash table where active session objects reside. By finding the right session, messages to it can be delivered. However, the graph contains no sessions at start-up, they are created later at run-time.

Both Muxes share an IsakmpFactory object which connects to their sideB0. This sideB0 is a sideB object with a slightly modified name, aiding the message routing when no session is found. Thus the Factory is connected from its both sides to a Mux. Figure 10 shows the basic setup of an ISAKMP server.

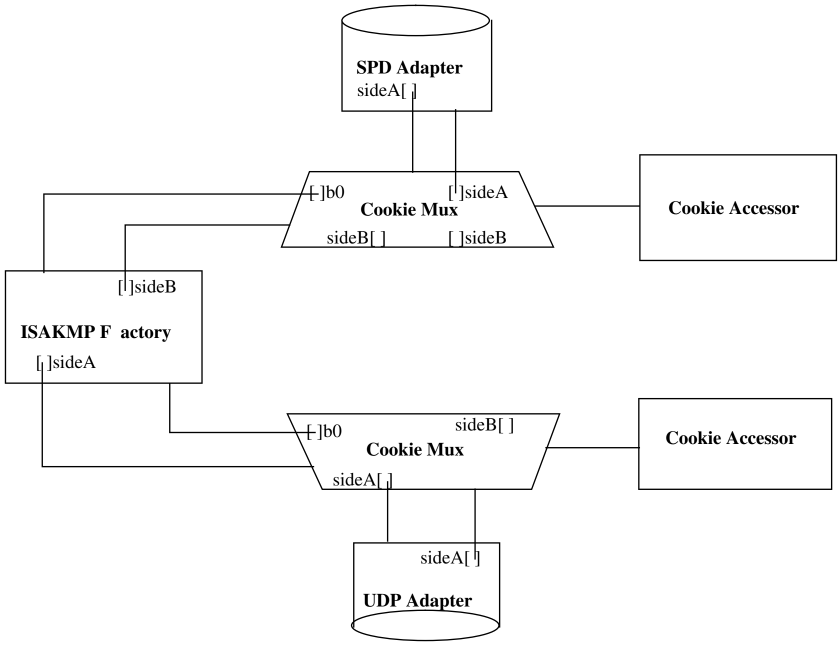

FIGURE 10. ISAKMP Conduits+ framework at start-up

The appearance of the graph changes when messages start to move in it. Let's assume the following situation: The IPSEC module performs an AH transform for outgoing messages and thus needs a Security Association for it. As this is the first message, the Cookie Mux attached to the SPDAdapter cannot receive a cookie from its Accessor, as the message contains no cookie. The Accessor however creates a cookie which is used in the outgoing message.

As no session is found matching the cookie, the Mux passes the message along to the ISAKMPFactory. The Factory creates a new Session object, which is attached between the two Muxes. Now the Session can receive the message, and create a new outgoing message. This same procedure happens every time there is no session the message can be routed to.

As the phase1 negotiation is completed, the Conduits graph needs to be modified to receive requests for creation of additional Security Associations . The graph evolves to the format shown in figure 11.

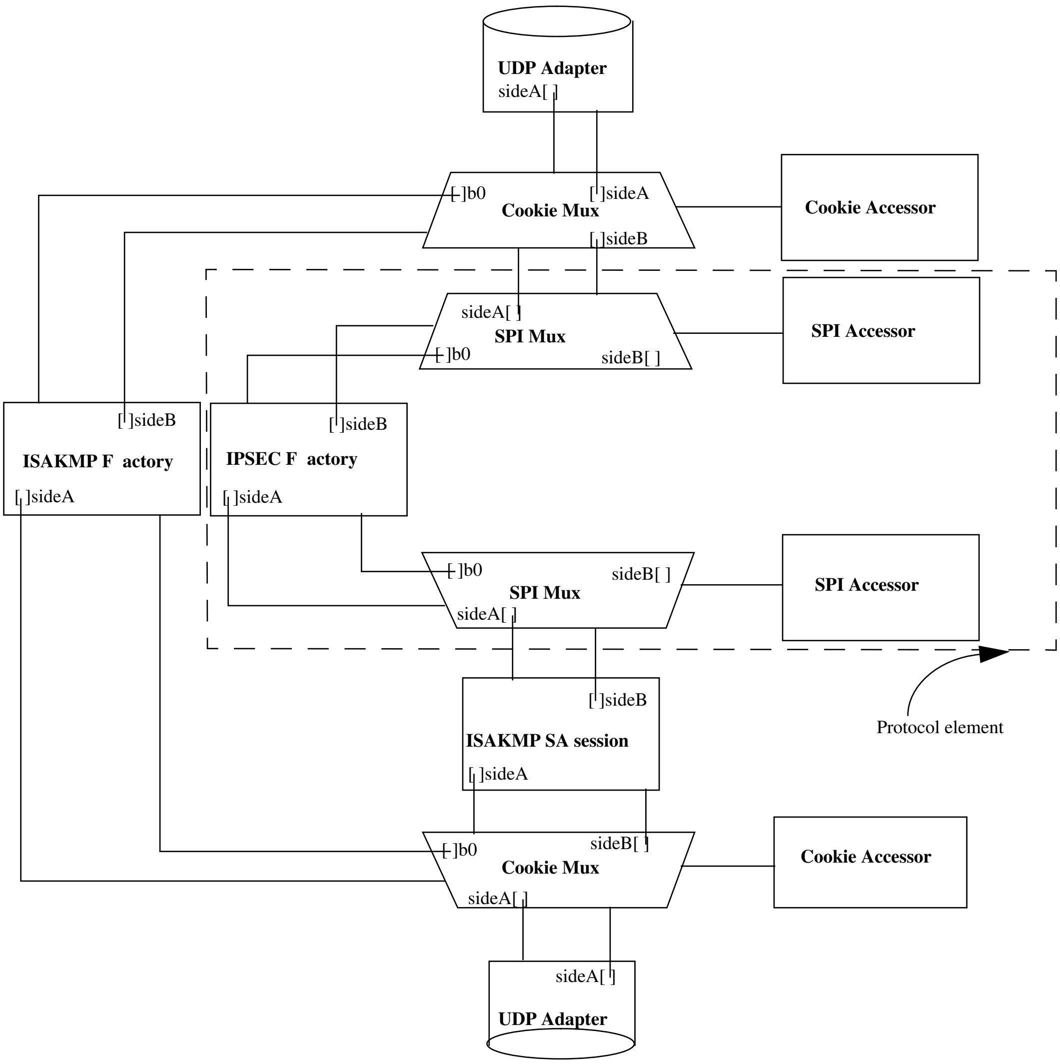

FIGURE 11. Framework ready to receive requests for creation of additional Security Associations

One w ay to accomplish this insertion to the graph is to ha ve a Protocol object that contains all the conduit elements needed to be inserted after phase1. The whole Protocol is inserted into the graph and is connected at the upper end to the Cookie Mux and at the lower end to the ISAKMP SA session. The Protocol element is shown inside the dashed line in figure 11. Before the insertion, the attachment of the Cookie Mux to the ISAKMP SA session must be broken. The upper Cookie Mux can still have other ISAKMP SA sessions connected to it because many such session can exist at the same time that have not yet completed the phase1 negotiation. A SPI value is passed to the SPD protocol that uniquely identifies the ISAKMP SA just created.

Now the SPD protocol can start the creation of an SA for the IPSEC AH protocol. The SPD module passes the SPI value that identifies the ISAKMP SA needed to secure the creation of an additional SA. The SPI value is used to route the message to the upper SPI Mux. The SPI value returned from the SPI Accessor does not match any session below the SPI Mux, so a new session has to be created. The IPSEC-Factory clones a prototype session which is installed in between the two SPI Muxes. Now the message can be routed to the appropriate session, which can create the initial outgoing message for phase2.

The new IPSEC session must consult its corresponding ISAKMP SA session for the algorithms and keys needed for securing the communication. Figure 12 shows the graph after the AH session creation. The figure shows only one session. The graph could contain an arbitrary number of ISAKMP SA sessions, which each can have a number of AH or ESP sessions above it. This because the protocol allows multiple phase2 negotiations to take place over one phase1 negotiation.

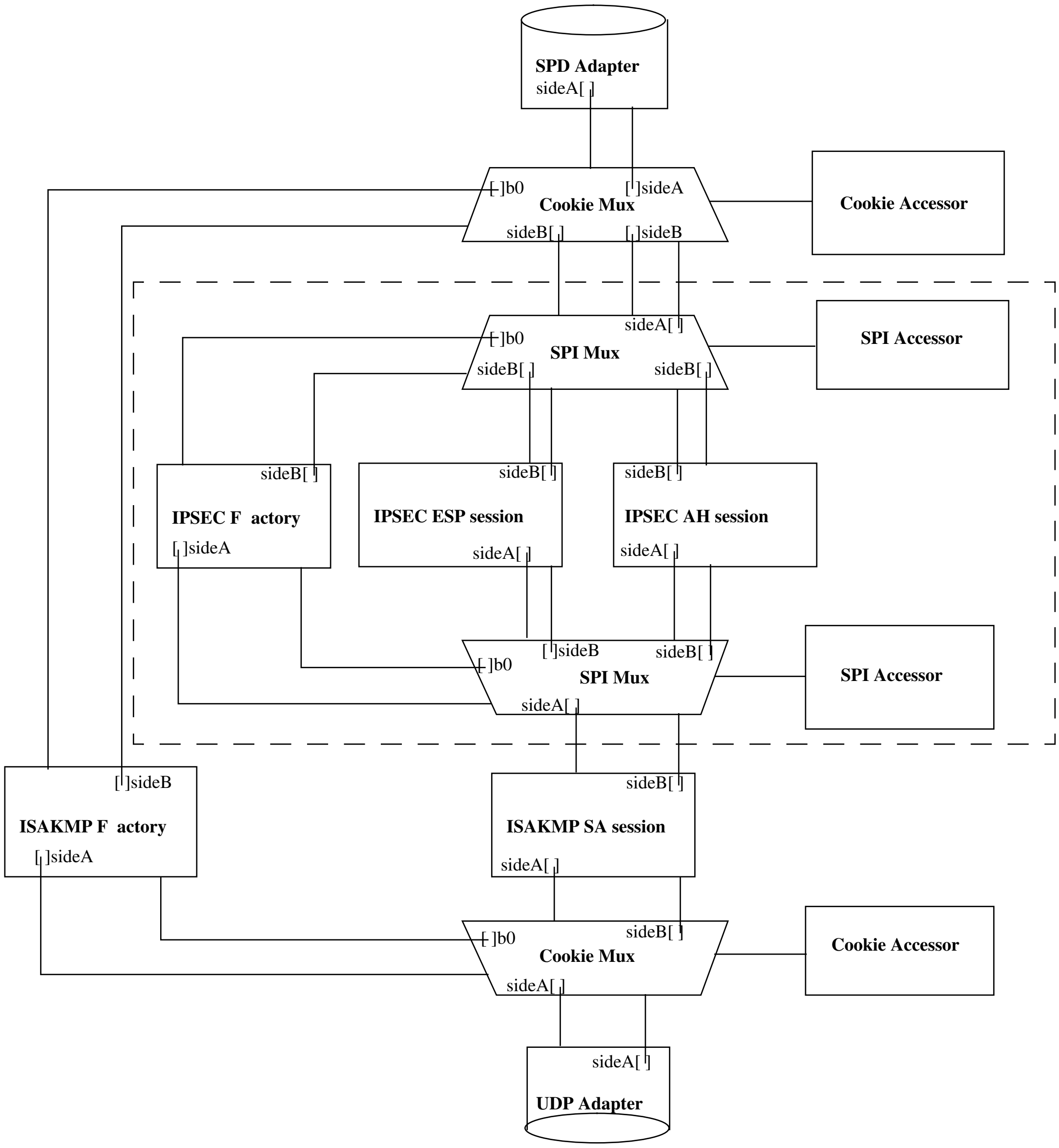

FIGURE 12. ISAKMP Conduits+ framework with sessions for ISAKMP SA and IPSEC AH, ESP creation

5.3 Class Model of Implementation

This section shows which classes are needed to create the Conduits+ framework, as well as other classes needed to implement the protocol functionality. The figures can be divided into two parts: one that shows the classes containing the Conduits objects and one that shows the Visitor object. The next section discusses how these classes are used and what they should perform.

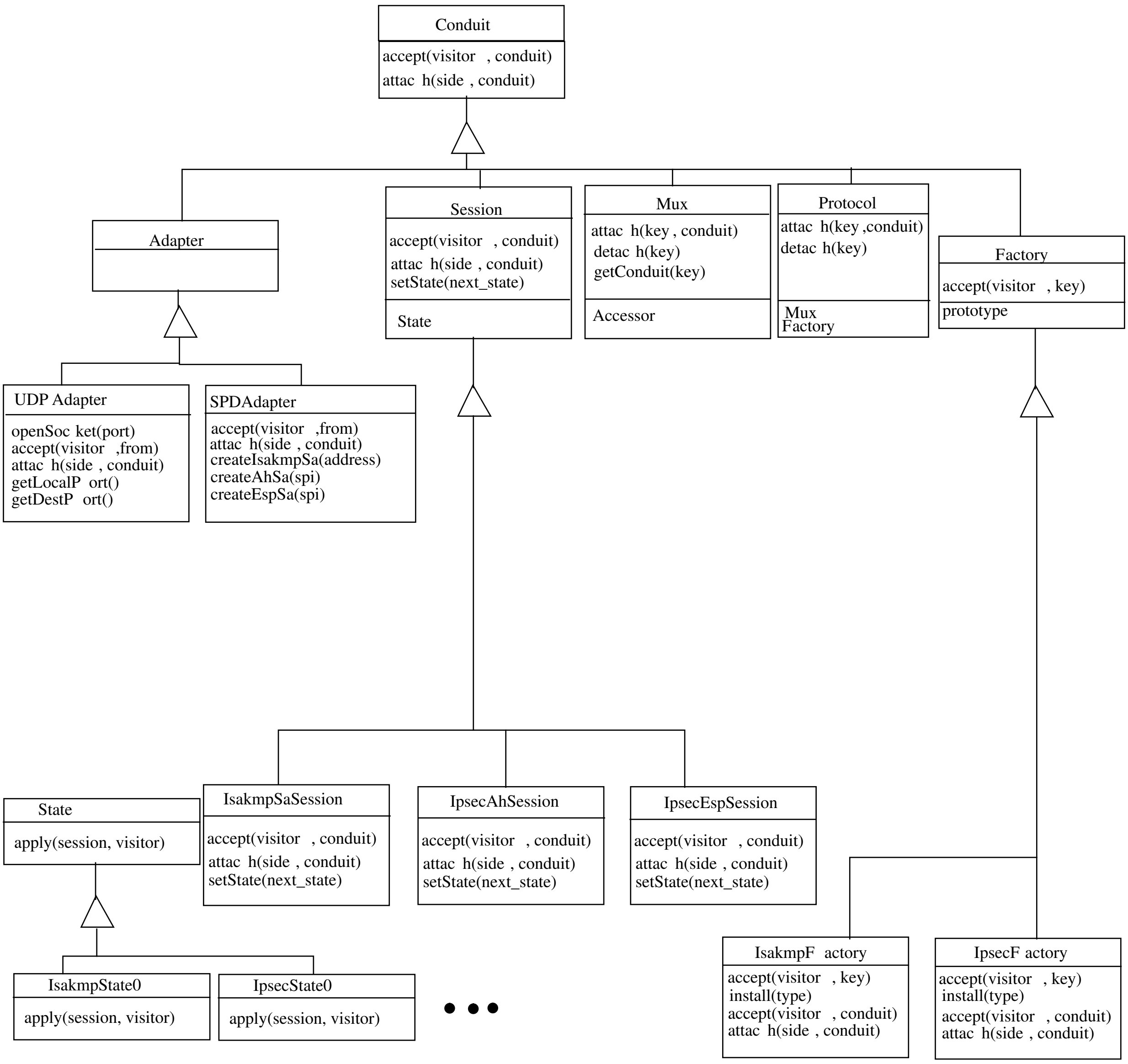

FIGURE 13. The Conduit class description

At the top of the graph resides the Conduit class which is one of the two elements defined in the Conduits+ framework. There are five types of conduits: Adapter, Session, Protocol, Mux, and Factory. These are all children to the parent class Conduit. In this implementation the Adapter class gets two children of its own: UDPAdapter and SPD-Adapter.

Three different Session classes are needed, one for ISAKMP itself and two for IPSEC. Each session contains the states it can be in. The State object is responsible for performing the protocol specific tasks. It does this through its apply method, where it gets the Session as one of its parameters.

The Protocol class contains the elements needed to be inserted into the graph after a phase1 negotiation. In the case of ISAKMP only two elements, the Mux and the Factory, are needed. The Protocol also needs methods for attaching itself into the graph, as well as methods to connect the Muxes and the Factorys together inside the Protocol. The benefit of a Protocol class is its simplicity. Only one insertion and deletion is needed instead of several ones.

Message multiplexing in the graph is done by a Mux class which contain the methods for routing messages. Each Mux contains its own Accessor.

To be able to create new elements, two Factories are needed. One that can clone ISAKMP Sessions and one that can clone both IPSEC AH- and IPSEC ESP sessions.

Figure 14 shows the class diagram of the Visitor element. There are two types of Visitors: Installers and MessageTransporters. The MessageTransporter carries payloads within it. These payloads contain protocol specific data. The MessageTransporter is responsible for transporting itself around the graph, and doing various tasks depending on its location. The Payload class has several children, namely all the payloads defined in the ISAKMP protocol.

The Installer has the responsibility to install the Sessions c loned by factories . The Installers at methods has different functionality than the MessageTransporters corresponding methods . They mainly attac h sessions to existing conduit elements .

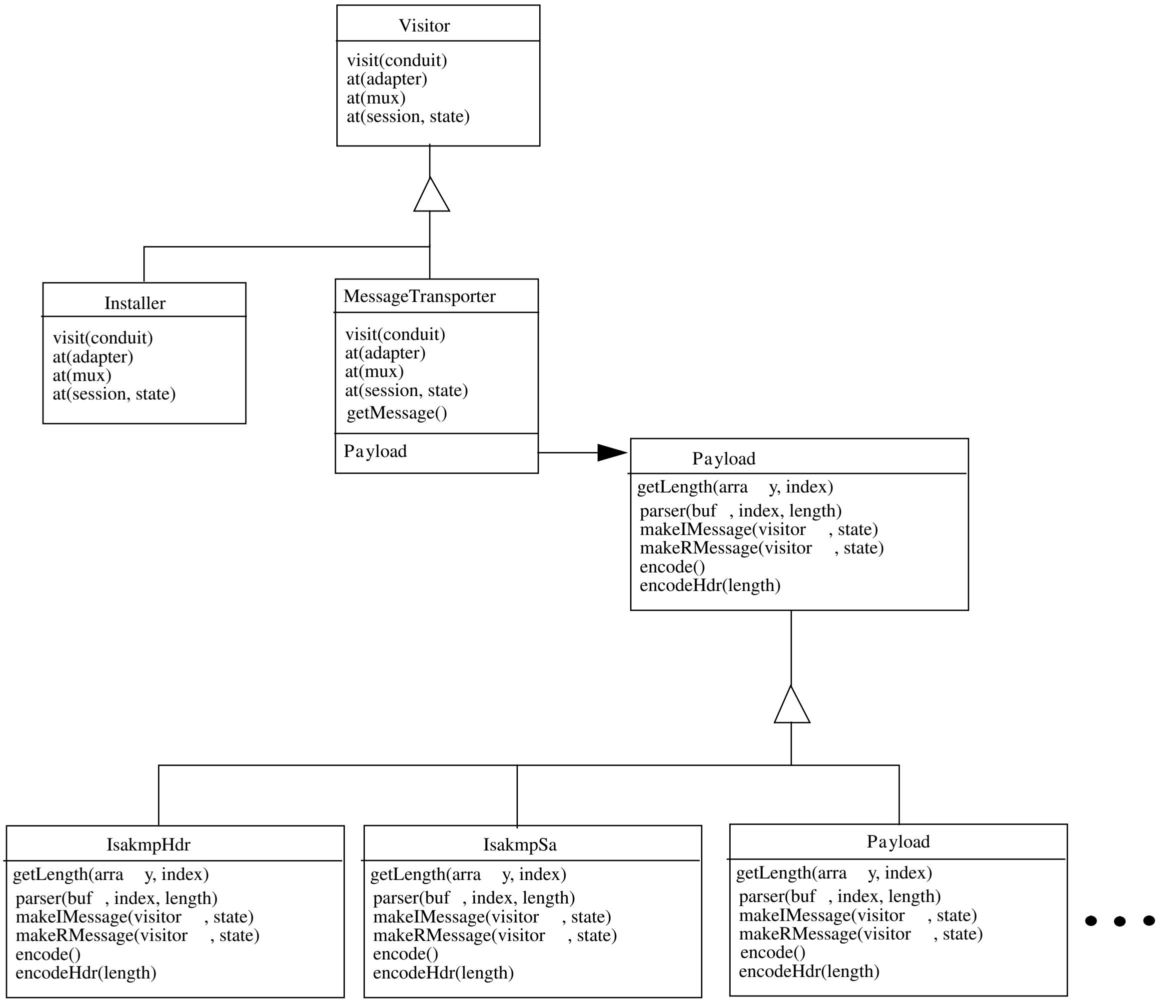

FIGURE 14. The Visitor class description

5.4 ISAKMP Functionality

This section describes in more detail how the ISAKMP server is implemented using the Conduits+ framework. Message Sequence Charts show the basic functionality in various parts of the graph. Figure 15 describes what happens when a message arrives at the UDP-Adapter. When a message arrives, a new MessageTransporter is created to carry the message around the graph. Next the message is opened or decoded from the incoming byte array. The result is a number of Payloads that contain data extracted from the message. These Payloads and their content are defined in the ISAKMP draft and described in section 3.5. The MessageTransporter and the data it contains are placed in a queue containing MessageTransporters waiting to be processed.

A ControlThread looping forever gets a notification from the Queue object that a new MessageTransporter has arrived in the queue. The thread gets the transporter from the queue. The MessageTransporter runs under this thread until it has accomplished its task. The MessageTransporter class contains a *visit* method which knows the MessageTransporters current location. This method knows where the MessageTransporter should go next and calls the appropriate Conduits *accept* method. As shown in figure 12, the MessageTransporter calls the UDP Adapters accept method.

These *accept* methods return a value to the *visit* method that tells the next element to visit. This is based on the sideA and sideB notion. All Conduits elements have other Conduits elements attached to their sides, and knows from which side the transporter was accepted into the Conduit. The framework says that if a Visitor arrives at sideA, it should leave at sideB. The *accept* method thus returns either the sideA or the sideB Conduit to the *visit* method depending on the side from where the Visitor was accepted. Now the *visit* method knows which Conduits *accept* method to call next. However, the *accept* method does not instantly return the next Conduit, but calls the Visitors *at* method instead. This method gets as its parameter the Conduit where it has arrived. It is then up to the Visitor, if it wants to perform some tasks at this Conduit, or just to move along.

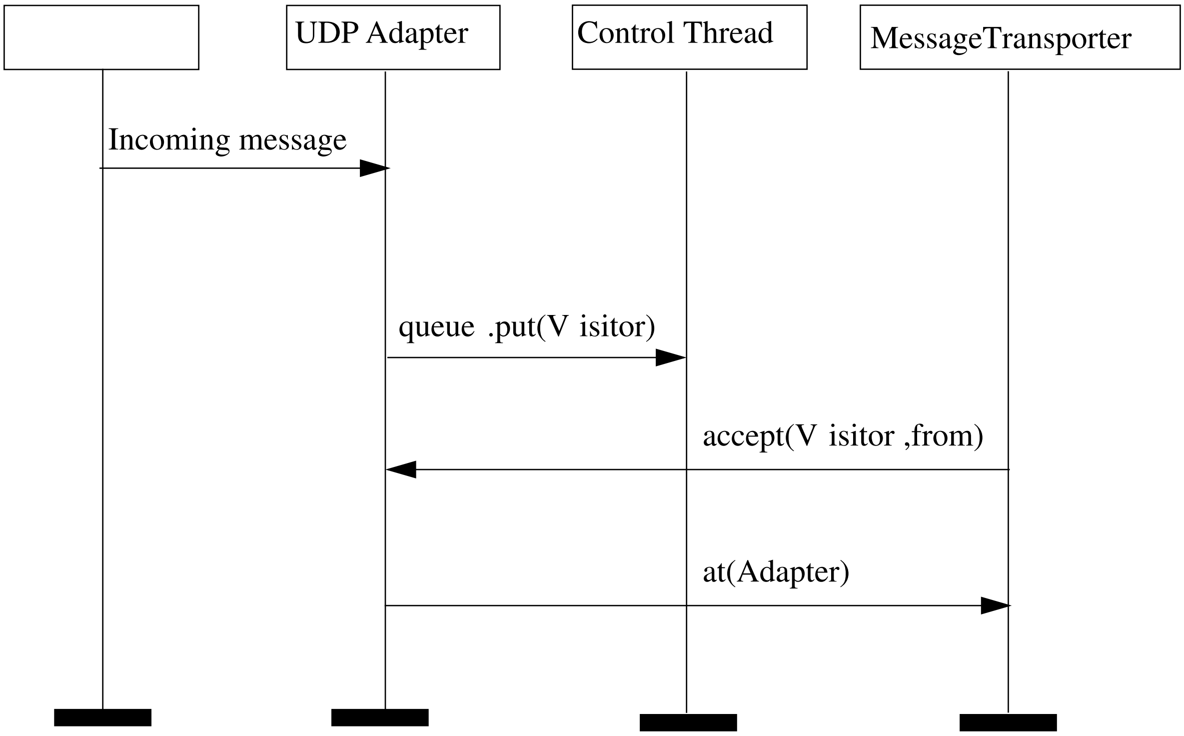

FIGURE 15. Message arrives at UDPAdapter

Sometimes it can happen that a Conduit the message should be routed to does not exist. Therefore each Mux has a Factory connected on its sideB0. If no Session matching the key returned by the Accessor is found, the Mux returns sideB0 as the next element to the *visit* method. In this way the Visitor calls the Factory's *accept* method.

As can be seen in figure 16, the Factory gets a Prototype of the Session it should create. It then creates an Installer and calls directly its *visit* method. As the Installer reaches the Mux, it attaches it to the newly created Session. The Factory also makes the Installer visit the other Mux connected on its sideB. These connections are shown in figures in chapter 5.2.

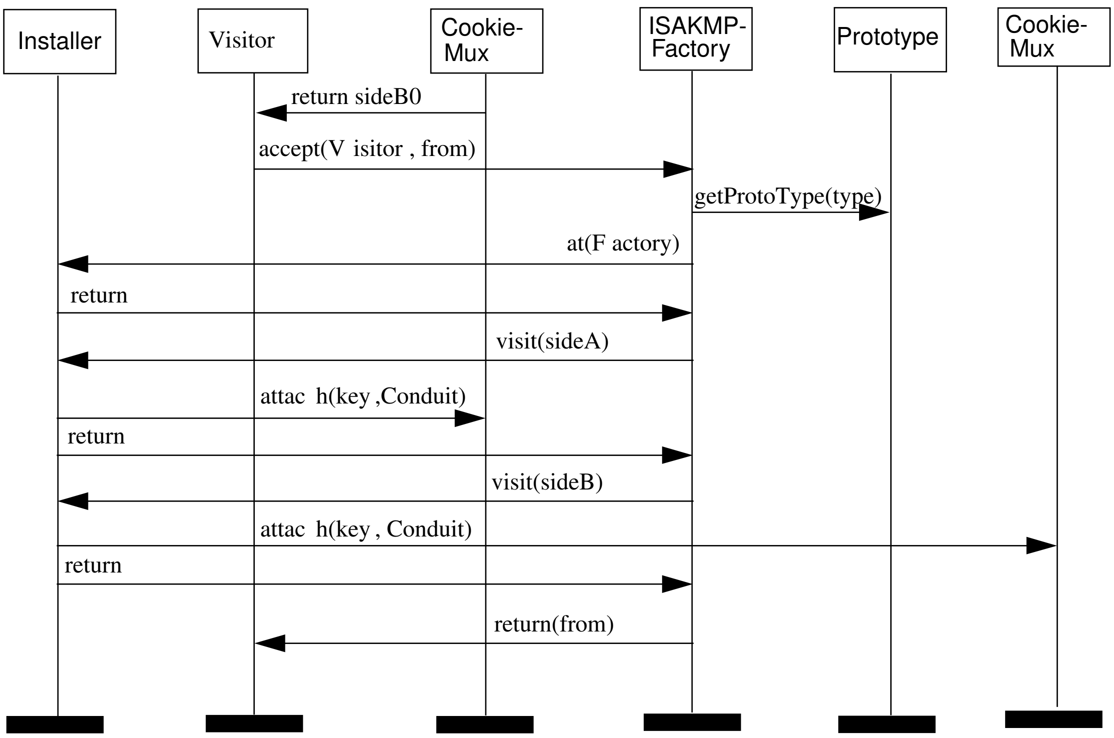

FIGURE 16. Creation of new Session

As all the connections are done, the Factory routes the MessageTransporter back to the Mux it came from, by returning the appropriate side. Now the transporter has a session to visit.

Figure 17 shows in more detail how the Visitor finds its way to the Session. The Visitor calls the *accept* method of the Mux which responds to the Visitor by calling its *at* method. The Visitor does not need to perform any tasks at the Mux, so the *at* method just returns. The Mux sends the data the Visitor carries to the Accessor. The Accessor finds the key from that message and returns it to the Mux. The Mux stores every Session it is attached to into a hash table. From this table a session can be found by using the key. The method *getSession* is used for this purpose.

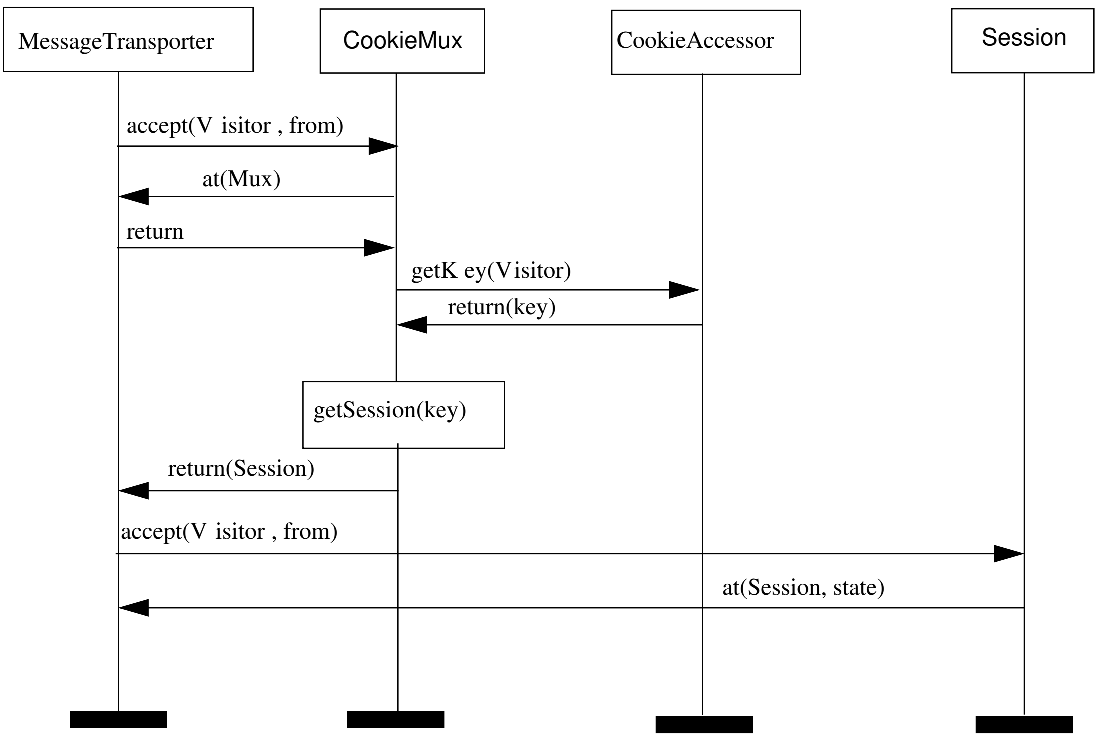

FIGURE 17. Message traversal in Conduit graph

When the right session is found, the Visitor can call the Sessions *accept* method and move along the graph.

The Session in itself does not perform any protocol functionality. It provides a method for accepting Visitors and a method for changing its current state. All other functionality resides in the State objects. All parameters that the Session needs must reside in the Session object. This because the States are implemented by using the Singleton pattern where all Sessions share a single instance of these States. In the case of an ISAKMP session this means that all data that has to do with SA parameters must reside in the session.

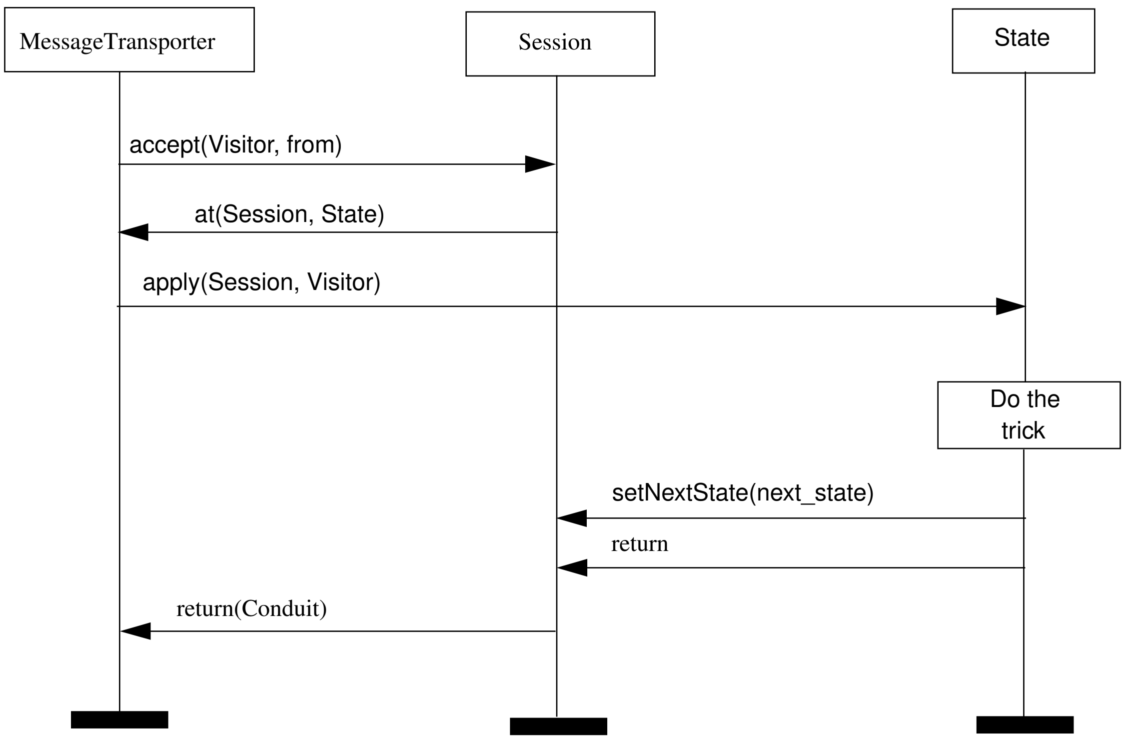

FIGURE 18. Functionality at State

Figure 18 shows how the Visitor calls the State object’s *apply* method. This method is responsible for doing all the tasks specified in the protocol documentation. Such tasks are to verify the correctness of an incoming message, and what actions should be taken when an error is found. The ISAKMP draft specifies on a payload basis what should be verified in a message.

The State also knows what kind of a message to create in response to an incoming message. The IsakmpMessage class contains methods for encoding messages and including the right payloads into it. These methods work in conjunction with methods found in every Payload class. The Payload itself is responsible for assigning proper values to its variables depending on the current state. Actually, there are two methods, one for assigning the variable values, and one for reading those values into a byte array. The IsakmpMessage object then collects all the byte arrays together to form a new outgoing message. Similarly the Payload is responsible for decoding itself from an incoming message.

The implementation relies on various cryptographic libraries to be able to perform all the security services needed. A library that can handle large numbers, such as a 1024 bit RSA key must be used because the largest variable type Java supports is 64 bits in length. This BigInteger class is used when computing Diffie-Hellman key values, and when dealing with RSA keys. The RSA keys are needed to sign messages, and for verification of the same message at the other end. As this implementation has no Key Distribution Center (KDC) to rely on, the RSA keys for a host is computed at start-up. The computation is based on a Java RSA library which uses native C functions to perform this. The RsaKeyHolder class can return both the public and private key of a host.

To compute a hash of a message, the implementation uses the Secure Hash Algorithm (SHA). The Data Encryption Standard (DES) is used to encrypt a message content when needed. Both of these are found in the cryptographic library.

6 Results and Experiences

The implementation of a protocol that is described in an Internet draft is not always easy. Especially ISAKMP, which is a fairly new protocol, is constantly changing. During this period of six months, three new drafts have been published. At the beginning of the project it was not even clear if ISAKMP would be the Internet key management protocol to be used.

The goals of partially implementing a key management protocol and testing the suitability of a communication protocol framework to aid this work have been interesting challenges. Using a new programming language like Java for the implementation makes it even more so. A time period of six months for achieving all this with no prior experience of frameworks or protocol implementation results in solutions that could be more elegant and effective. Further development of the prototype adds these elements as well.

6.1 Conduits+ Suitability

To design and implement a good working framework requires careful planning. The framework uses a number of design patterns and becomes quite a big totality. As reusable object oriented software is hard to write, it makes the framework hard to get right the first, or even the second time. Prior experience in designing and implementing frameworks is of great help. When planning a framework for the first time, one tends to add too much functionality to it, at the cost of generality. It is hard to keep out functionality that does belong to a protocol specific part, but not to the framework.

Overall, if one succeeds in making the framework generic to a degree where most of the implementation needs to be done in the session and state objects, it is quite suitable for implementing protocols. The biggest problems arise in the need to make changes to the Conduit graph. If the framework can cope with any type of changes to the graph, it is highly reusable.

Because the state object performs all the protocol specific parts, the framework is better suited for a protocol that has clear states, and can be implemented as a finite state machine. This is the case with the ISAKMP protocol, which fits fine in the Conduits+ framework model.

6.2 Java Suitability

Basically Java can be used for writing communication protocols. It is easy to get started in writing programs with Java. It offers many libraries that supports the implementation. The Java networking library contains easy to use methods for dealing with communication over connection oriented- and connectionless socket interfaces. This makes it easy to create a server running on top of the TCP/IP stack, which is the case with ISAKMP.

Portability is no problem because the program can be run on any platfrom. Speed on the other hand can cause problems in implementations that are time critical. Such functionality can be implemented in a native language, e.g. in the C programming language. Java makes it possible to call these C functions from Java. It requires some time to get used to, but is a quite straightforward process after that. One confusing property is that Java primitive data types do not map directly to C data types. Java does not support e.g. unsigned chars which are used in communication software implemented in C.

Libraries for dealing with large numbers and cryptographic functions are not supported in early Java software packets. However, such packets exist, and can easily be included.

7 Prototype Evaluation and Conclusions

The goal of this study was to implement a part of the Internet Security Association and Key Management Protocol. The key generation, and key transfer is handled as specified in the Oakley protocol. The implementation was based on the draft: [ISAOAK97]. The implementation was limited to contain functionality for creation of Security Associations for both ISAKMP and other security protocol use.

The subpart of ISAKMP and Oakley protocols were implemented using a Conduits+ framework as a skeleton. The framework provides for the ability to make changes to the protocol as the drafts are updated, without modifying the framework. This because the protocol functionality is implemented in the state object of the protocol.

The implementation can easily be updated as new techniques are developed to break encryption algorithms or stronger security mechanisms are developed. ISAKMP is developed to be algorithm and key exchange independent, and the framework is not dependent on these parts either.

The use of an object-oriented approach to the design simplifies the task of modifying the framework at run-time. The modification is achieved by either using inheritance or composition. When using inheritance the framework provides for base classes that define common functionality of its subclasses. The subclasses can then redefine some functionality and use some of the functionality the base class provides. Composition is achieved by creating instances of existing objects, and by using the methods they provide. By using composition the user of the framework does not exactly need to know every detail of how things are done. It is sufficient to know what a method does, and how it is called. This is the black-box approach to designing frameworks.

Even if Conduits+ strives to be a black-box framework it uses much inheritance. Another implementation that wants to use the framework needs to define own subclasses, and thus needs to know the framework implementation quite well. For example, ISAKMP could be implemented over TCP instead of UDP, so a new TCP Adapter class should be inherited from the Adapter class.

This implementation can negotiate security parameters to both ISAKMP itself and for a Security Policy Daemon. The SPD is responsible for initializing Security Association creations with the right parameters and for checking the validity of incoming requests.

Basically the implementation provides for two interfaces. Both of these are placed in Adapters connecting the framework with the world outside. The UdpAdater sends data to, and receives data from the network, while the SpdAdapter provides for a group of methods for negotiation of Security Associations.

Performance measures were not inserted in the project plan but observing the protocol run with bare eyes gives the impression that more speed is required. Especially the computation of cryptographic keys takes a long time. Of course these keys are generated only seldom and a "real" implementation would fetch e.g. a public RSA algorithm from a Key Distribution Center.

8 References

[ATK95a] Atkinson, R., Security Architecture for Internet Protocol, RFC 1825, Naval Research Laboratory, 1995

[ATK95b] Atkinson, R., IP Authentication Header, RFC 1826, Naval Research Laboratory, 1995

[AZI96] Azis, A., SKIP extension for Perfect Forward Secrecy (PFS), Work in Progress, Sun Microsystems, Inc., February, 1996

[BEL89] Bellowin, S., Security Problems in the TCP/IP Protocol Suite, ACM Computer Communication Review, Vol. 19, No. 2, April 1989, pp. 32-48

[BEL96] Bellowin, S., Problem Areas for the IP Security Protocols, Work in Progress, AT&T Research, 1996

[BLA96] Blaze, M., Whitfield, D., Rivest, R., Schneier, B., Shimomura, T., Thompson, E., Wiener, M., Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security, A Report by an Ad Hoc Group of Cryptographers and Computer Scientist, January 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

[COND+] Engel, R., Hüni, H., Johnson, R., A Framework for Network Pro tocol Software. Technical report, ACM, 1995.

[FLAN] Flanagan, D., Java in a Nutshell, O'Reilly & Associates Inc, 1996

[GAMMA] Gamma, E.,Helm, R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[GLI83] Gligor, V., A Note on the Denial of Service Problem, Proceedings of the IEEE Symposium on Security and Privacy, 1983

[GOS] Arnold, K., Gosling, J., The Java Programming Language, Addison-Wesley, 1996

[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.

[ISAKMP97] Maughan, D., Schertler, M., Schneider, M., Turner, J., Internet Security Association and Key Management Protocol (ISAKMP), version 7, draft-ietf-ipsec-isakmp-07.{ps, txt}.

[ISAOAK97] Harkins, D., Carrel, D., The resolution of ISAKMP with Oakley, version 3, draft-ietf-ipsec-isakmp-oakley-03.txt.

[JAVA] Sun Microsystems, The Java Language: A White Paper

[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

[KS96a] Karn, P., and W. Simpson, The Photuris Session Key Management Protocol, Work in Progress draft-ietf-ipsec-photuris-09, Qualcomm, Inc., Daydreamer, February 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.

[NBS93] National Bureau of Standards. Secure Hash Standard, May 1993, Federal Information Processing Standards Publication 180.

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

[SCH96] Schneier, B., Applied Cryptography, Second Edition, John Wiley & Sons, New York, 1996.

[SIM95] W. Simpson, Photuris Extensions, Work in Progress, Daydreamer, November 1995.

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

[STE90] Stevens, R., UNIX Network Programming, Prentice Hall, New jersey, 1990

[TAA96] Aalto, T. P., A Unix Streams Implementation of the Internet Protocol Security, Helsinki University of Technology, September 1996