

IPsec Intrusion Detection Analysis:
Using data from an Ericsson Ethernet Interface Board

Master Thesis Report

20 March 2008

Achille Faienza
and
Julian Amso

KTH Examiner and Supervisor
Professor Gerald Q. Maguire Jr.

Ericsson Supervisors
Karl Knutsson
Sheng-Chou Li

Not the official thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

Not the official thesis.

PREFACE

This report is part of the final examination for the Master of Science in Engineering program, Civilingenjörsprogrammet, at Royal Institute of Technology (KTH) in Stockholm, Sweden. The project was conducted at Ericsson AB in Älvsjö, IP Design department, FTP/DRX section during year 2007 and 2008. The supervisor and examiner at KTH was Professor Gerald Q. Maguire Jr. at Department of Communication Systems (CoS), School of Information and Communication Technology (ICT), KTH. Our supervisors at Ericsson AB were Karl Knutsson and Sheng-Chou Li.

ABSTRACT

IP security (IPsec) is commonly used for protection in Virtual Private Networks (VPN). It is also used for the protection of traffic between nodes in third generation (3G) mobile networks. The main duty of telecommunication operators is to assure the quality of service and availability of the network for their users. Therefore knowledge of threats that could affect these requirements is of relevance. Denial of Service (DoS) and other attacks could constitute serious threats in 3G networks and, if successful, they could lead to financial and reputation damage for the telecommunication operator. One of the goals of each telecommunications vendor is to produce equipment and software in such a way as to reduce the risk of successful attacks upon networks built using their equipment and software. This master's thesis aims to identify the classes of attacks that could affect the regular operation of an IPsec-protected network. Therefore, the IPsec protocol and its possible weaknesses are explained. As practical demonstration of these ideas, an Intrusion Detection Analyzer prototype for an Ericsson Ethernet Interface board was developed to detect anomalous IPsec-protected traffic.

SAMMANFATTNING

IP Security (IPsec) protokollet används bl.a. för att skydda Virtuellt Privat Nätverk (VPN). Protokollet används även för att skydda noderna i tredje generationens (3G) mobila nätverk. Telekomoperatörernas uppgift går bl.a. ut på att se till att de mobila näten är tillgängliga för användarna samt garantera en viss garanterad tjänstekvalitet. Därför är kunskapen om de olika hoten som påverkar dessa faktorer relevant. Överbelastningsattacker och andra attacker kan utgöra ett stort hot mot bl.a. 3G nät. Om dessa attacker lyckas kan de leda till finansiella skador och ett skadat anseende för telekomoperatörerna. Ett av målen för telekomtillverkarna är att tillverka produkter och program som kan minimera riskerna för en attack och skadorna som åstadkoms på ett nätverk uppbyggt med deras utrustning. Detta examensarbete har som mål att identifiera de olika typer av attacker som kan påverka driften av IPsec-skyddade nätverk. IPsec-protokollet och dess svagheter är förklarade. Svagheter och problem med vissa implementationer nämns också. I detta arbete ingår också att utveckla en Intrusion Detection Analyzer prototyp för ett Ericssons Ethernet Gränssnitt kort för att upptäcka avvikande IPsec-skyddad trafik.

ACKNOWLEDGMENT

We would start with expressing the sincerest gratitude towards our KTH supervisor, Professor Gerald Q. "Chip" Maguire Jr., for the valuable advices and help during the thesis project, through hi-tech comments and corrections and real-time email replies.

We would also like to express our gratitude towards Ericsson AB, in particular to Sheng-Chou Li at Ericsson Älvsjö and Daniel Flemström at Mälardalens högskola, for offering us the opportunity to work on an interesting topic for the Master's Thesis project. We are grateful for the help and advices received from our Ericsson supervisor, Karl Knutsson. We would also like to thank Sven Stenström at Ericsson AB for his help and clear explanations about the Ericsson Ethernet Interface board used. We are grateful for the help and advices received from our KTH colleague, Daniel Jaurén.

Julian

Finally, I would like express my gratitude towards my mother, father and brother for their support and care.

Achille

I would like to thank my supervisor at Politecnico di Torino, Professor Marco Mellia. Last but not least, I would like to thank my family for their continuous support.

Stockholm, March 2008

TABLE OF CONTENTS

Preface	i
Abstract	ii
Sammanfattning	iii
Acknowledgment	iv
List of figures	viii
List of tables	xi
Acronyms and Abbreviations	xii
1 Introduction	1
1.1 Problem Statement	1
1.2 Ericsson Platforms	1
1.3 Ericsson Ethernet Interface board	2
<i>1.3.1 Network Processor</i>	4
<i>1.3.2 IPsec processing in Ericsson Ethernet Interface board</i>	5
1.4 Intrusion Detection.....	8
2 Background	12
2.1 The Internet Protocol	12
2.2 IP Security.....	14
2.2.1 <i>Security Association</i>	15
2.2.2 <i>Security modes</i>	18
2.2.3 <i>Security protocols</i>	19
2.2.4 <i>Authentication algorithms</i>	25
2.2.5 <i>Encryption algorithms</i>	26
2.2.6 <i>Key Management</i>	27

2.2.7	<i>Implementations and Architectures</i>	27
2.3	Related work	28
3	Criticism against and weaknesses in IPsec	30
3.1	Ferguson and Schneier evaluation.....	30
3.2	Probable plaintext in IPsec.....	31
3.2.1	<i>Probable Plaintext in the IP Header</i>	31
3.2.2	<i>Probable Plaintext in the TCP header and UDP header</i>	32
4	Attacks against IPsec	33
4.1	Replay attack.....	33
4.2	CPU overload DoS attack	33
4.3	Sliding Window Attack.....	34
4.4	Attacks against unauthenticated ESP traffic in CBC mode.....	35
4.4.1	<i>Bellovin's attack</i>	35
4.4.2	<i>Paterson's and Yau's attack</i>	36
4.5	IPsec attacks summary	39
5	Method	41
5.1	Different scenarios of IPsec processing	42
5.1.1	<i>SA lookup failure</i>	43
5.1.2	<i>IPsec processing failure</i>	44
5.1.3	<i>Security policy violation</i>	45
5.1.4	<i>Correct IPsec packet</i>	45
5.2	LSI System Performance Analyzer.....	46
5.2.1	<i>Classification of modified IPsec packets</i>	47
5.2.2	<i>Traffic Management of modified IPsec packets</i>	48
5.2.3	<i>Creating a SED Script in the SPA</i>	50
5.2.4	<i>Creating a DID in the SPA</i>	50
5.3	Software IPsec stack implementation	51
5.3.1	<i>Traffic Generator</i>	51

5.3.2	<i>Analyzer program</i>	52
5.3.3	<i>Monitoring IPsec traffic</i>	55
5.3.4	<i>The Sniffer</i>	57
6	Analysis	58
6.1	Simulation in the LSI System Performance Analyzer.....	58
6.1.1	<i>Capturing IPsec packets</i>	58
6.1.2	<i>Reinjecting modified IPsec packets</i>	58
6.1.3	<i>Forwarding modified IPsec packets to the Analyzer</i>	59
6.2	Simulation using a software IPsec stack implementation	60
6.2.1	<i>Simulation of attack at SA lookup phase</i>	64
6.2.2	<i>Simulation of attack at IPsec processing phase</i>	67
6.2.3	<i>Simulation of attack at Security Policy verification phase</i>	71
7	Conclusions	77
7.1	Conclusion	77
7.2	Future work	78
	REFERENCES	79
	Appendix A	84
	Appendix B	87
	Appendix C	89
	Appendix D	90
	Appendix E	91

LIST OF FIGURES

Figure 1: Ericsson Layered Platform Structure [14]	2
Figure 2: Main modules of the Ericsson Ethernet Interface board.....	3
Figure 3: Main blocks of the APP300 Network processor series.....	5
Figure 4: SPP Result Header.....	5
Figure 5: Position of Fatal Errors bits in SPP Result Header.....	6
Figure 6: Position of Non Fatal Errors bits in SPP Result Header.....	7
Figure 7: IDS placed before an IPsec gateway	10
Figure 8: IDS placed after an IPsec gateway	10
Figure 9: IDS placed in direct connection with IPsec gateway.....	10
Figure 10: A matrix illustrating different IDS outcomes	11
Figure 11: Format of IPv4 Datagram.....	12
Figure 12: Format of an IP datagram showing transport mode and tunnel mode in comparison to the original packet	19
Figure 13: AH Header.....	20
Figure 14: Format of IPv4 datagram protected with IPsec AH Transport mode	21
Figure 15: IPv4 Datagram Protected with IPsec AH Tunnel Mode.....	22
Figure 16: ESP packet.....	23
Figure 17: IPv4 Datagram Protected with IPsec ESP Transport Mode	24
Figure 18: IPv4 Datagram Protected with IPsec ESP Tunnel Mode.....	25
Figure 19: Modification of an IPsec packet in Bellovin's attack	36
Figure 20: Phase 1 modification of IPsec packet in Paterson's and Yau's attack [64]	37
Figure 21: Phase 2 modification of IPsec packet in Paterson's and Yau's attack [64]	38
Figure 22: A logical overview of the Intrusion Detection Setup for the Ericsson Ethernet Interface Board	42
Figure 23: The different steps for an incoming IPsec packet to traverse through an IPsec gateway	43
Figure 24: View of a network protected by two IPsec gateways	43

Figure 25: Screenshot of the LSI System Performance Analyzer.....	46
Figure 26: Chart of fields in the fTransmit() function	47
Figure 27: Only the indicated data of the IPsec packet is sent from the Classification Engine to the Traffic Manager in case of an IPsec processing failure (E2 error packet)	48
Figure 28: The prepended headers of the PDU sent to the Analyzer in case of an IPsec processing failure (E2 error packet)	48
Figure 29: Creating a Destination ID (DID) in the SPA	51
Figure 30: Screenshot of the packet traffic generator executing	52
Figure 31: An illustration of sliding window 1 to monitor E1 error packets	53
Figure 32: An illustration of sliding window 1 and 2 implementation to monitor E1 error packets.....	53
Figure 33: Different sliding window alert cases (No size or threshold value set, just for illustration)	55
Figure 34: Configuration in case of data analysis from one Ericsson Ethernet Interface board, using TAP	56
Figure 35: Configuration in case of data analysis from one Ericsson Ethernet Interface board, using port mirroring	56
Figure 36: Screenshot of IPsec reinjection in the SPA simulator	59
Figure 37: Screenshot of a modified IPsec packet with an IPsec block error set in the SPP result header in the SPA	60
Figure 38: Screenshot of the startup of Analyzer.....	61
Figure 39: Logical configuration for simulation.....	62
Figure 40: Screenshot of Tcpdump trace of traffic between Belkar and Elan	63
Figure 41: Configuration of the hosts in the lab and the flow of datagrams (traffic on interface wm0)....	64
Figure 42: Screenshot of programs running on Elan during SA lookup attack simulation (Sniffer)	65
Figure 43: Screenshot of programs running on Elan during SA lookup attack simulation (Script to filter logfile)	65
Figure 44: Screenshot of programs running on Elan during SA lookup attack simulation (SPD).....	65
Figure 45: Screenshot of programs running on Elan during SA lookup attack simulation (Tcpdump).....	66
Figure 46: Screenshot of programs running on Belkar during SA lookup attack simulation (Analyzer) ..	66
Figure 47: Screenshot of programs running on Belkar during SA lookup attack simulation (Attacker) ...	67
Figure 48: Screenshot of programs running on Belkar during SA lookup attack simulation (SAD and SPD).....	67

Figure 49: Screenshot of Netdude used for IPsec packet modification	68
Figure 50: Screenshot of programs running on Elan during IPsec processing failure (SPD)	68
Figure 51: Screenshot of programs running on Elan during IPsec processing failure (Script to filter logfile)	69
Figure 52: Screenshot of programs running on Elan during IPsec processing failure (Sniffer).....	69
Figure 53: Screenshot of programs running on Elan during IPsec processing failure (Tcpdump)	70
Figure 54: Screenshot of programs running on Belkar during IPsec processing failure (SPD and SAD) .	70
Figure 55: Screenshot of programs running on Belkar during IPsec processing failure (Analyzer).....	71
Figure 56: Screenshot of programs running on Belkar during IPsec processing failure (Attacker).....	71
Figure 57: Screenshot of Tcpdump trace of the traffic and SPD dump on Elan testing security policy enforcement (SPD)	72
Figure 58: Screenshot of Tcpdump trace of the traffic and SPD dump on Elan testing security policy enforcement (Tcpdump)	72
Figure 59: Tcpdump trace of the traffic on Belkar testing the faulty IPsec implementation on Elan	73
Figure 60: SPD dump on Belkar testing the faulty IPsec implementation on Elan	73
Figure 61: Screenshot of programs running on Elan during Security Policy violation (Tcpdump).....	74
Figure 62: Screenshot of programs running on Elan during Security Policy violation (SPD).....	74
Figure 63: Screenshot of programs running on Elan during Security Policy violation (Script to filter logfile)	74
Figure 64: Screenshot of programs running on Elan during Security Policy violation (sniffer).....	75
Figure 65: Screenshot of programs running on Belkar during Security Policy violation (SAD and SPD)	75
Figure 66: Screenshot of programs running on Belkar during Security Policy violation (Analyzer)	76
Figure 67: Screenshot of programs running on Belkar during Security Policy violation (Attacker).....	76

LIST OF TABLES

Table 1: The two most significant bits in the SPP result header indicating type of IPsec processing error.	6
Table 2: IPsec block Fatal Error Codes	7
Table 3: IPsec block Non-Fatal Error Codes	8
Table 4: Sample of Security Association	16
Table 5: Attack scenarios during SA lookup failure	44
Table 6: Attack scenarios during IPsec processing failure	44
Table 7: Attack scenarios during security policy violation	45
Table 8: The different packet types sent to the Analyzer.....	49
Table 9: The probabilities of different type of errors received from the board.....	54
Table 10: The different cases when the Analyzer program need to alert the operator.....	54

ACRONYMS AND ABBREVIATIONS

2G	Second Generation Mobile Networks
TDES	Triple DES
3G	Third Generation Mobile Networks
3GPP	The 3 rd Generation Partnership Project
AAA	Authentication, Authorization And Accounting
ACK	Acknowledgment
AES	Advanced Encryption Standard
AES-CTR	AES Counter Mode
AH	Authentication Header
ALG	Algorithm
APP	Advanced Payload Plus (LSI Processor Family)
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
BITS	Bump In The Stack
BITW	Bump In The Wire
BSC	Base Station Controller
BSD	Berkeley Software Distribution
CBC	Cipher Block Chaining
CDMA2000	Code Division Multiple Access 2000
CE	Classification Engine
C-NP	C For Network Processor
CoS	Class Of Service
CPP	Connectivity Packet Platform
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CTR	Counter
CVE	Common Vulnerabilities And Exposures
DES	Data Encryption Standard
DID	Destination ID
DMA	Direct Memory Access
DoS	Denial Of Service
DS	Differentiated Services
E1	E1 Error Packet (SA Lookup Failure)
E2	E2 Error Packet (Ipssec Processing Failure)

E3	E3 Error Pocket (Security Policy Violation)
ECB	Electronic Code Book
EDE	Encryption Decryption Encryption
EEE	Encryption Encryption Encryption
ESP	Encapsulated Security Payload
FE	Fatal Error
FIPS	Federal Information Processing Standard
FPL	Functional Programming Language
GCC	Gnu C Compiler / Gnu Compiler Collection
GSM	Global System For Mobile Communications
HLEN	Header Length
HLR	Home Location Register
HMAC	Hashed Message Authentication Code
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
ICV	Integrity Check Value
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange Protocol
IP	Internet Protocol
IPsec	IP Security
IPv4	IP Version 4
IPv6	IP Version 6
ISAKMP	Internet Security Association And Key Management Protocol
IV	Initialization Vector
LAN	Local Area Network
LIU	Line Interface Unit
MAC	Message Authentication Code
MAC	Media Access Control
MD5	Message Digest 5
MGCF	Media Gateway Control Function
MSC	Mobile Switching Center
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NE	Non Fatal Error
Netdude	Network Dump Data Displayer And Editor
NIST	National Institute Of Standards And Technology

NP	Network Processor
NPU	Network Processor Unit
NSA	National Security Agency
O&M	Operation & Maintenance
OSI	Open System Interface
PDSN	Packet Data Serving Nodes
PDU	Protocol Data Unit
PKI	Public Key Infrastructure
QoS	Quality Of Service
RBS	Radio Base Station
RFC	Request For Comments
RNC	Radio Network Controller
RSP	Route Switch Processor
SA	Security Association
SAD	Security Association Database
SED	Stream Editor
SHA	Secure Hash Algorithm
S/N	Signal To Noise Ratio
SPA	System Performance Analyzer
SPAN	Switch Port Analyzer
SPD	Security Policy Database
SPI	Security Parameter Index
SPI-3	System Packet Interface Level 3
SPP	Security Protocol Processor
STCP	Secure TCP
SW	Sliding Window
SYN	Synchronize
TAP	Test Access Port
TCP	Transmission Control Protocol
TM	Traffic Manager
TOS	Type Of Service
TSP	Telecom Server Platform
TTL	Time To Live
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
WCDMA	Wideband Code Division Multiple Access
VER	Version
VPN	Virtual Private Network

1 INTRODUCTION

1.1 Problem Statement

Internet Protocol (IP) is increasingly used to carry telecommunication traffic, an example of this which is specifically relevant to this thesis is the traffic between Radio Base Stations (RBS) and Radio Network Controllers (RNC). The 3rd Generation Partnership Project (3GPP), a collaborating group of telecommunication vendors & associations concerned with the evolution of GSM and its 3rd Generation (3G) mobile network specification, has specified that IPsec must be supported in order to provide IP network layer security [1]. Ericsson Ethernet Interface board, used to provide a modular interface to IP and ATM networks, is used in many mobile networks and because of the 3GPP requirements it will need to support **IPsec ESP in tunnel mode with authentication** in the future.

Although IPsec provides network security, specifically confidentiality & authentication, the protocol has been criticized for its complexity and because certain configurations could expose an implementation to serious attacks. Even if a secure configuration is used, a system protected by IPsec could still be vulnerable to some forms of attacks, i.e. Denial of Service (DoS) attacks. A DoS attack might reduce the capacity of a mobile network, affecting the quality of service of one or more calls and/or limit the number of simultaneous users. Traditional Intrusion Detection Systems (IDS) cannot detect these attacks, since this IDS is usually located after the IPsec gateway. If the IDS is located after the IPsec gateway, then attack traffic rejected and discarded during IPsec processing will not reach an IDS, hence the IDS will neither be able to generate alerts for attacks nor capture traffic for subsequent analysis. Thus while IPsec may be successful in preventing this traffic from crossing the gateway, the network still suffers from the loss in capacity due to the resources which have been utilized by this rejected traffic, and these attempts should be detected.

This thesis will survey known weaknesses and attacks against the IPsec protocol. It will also look in detail at attacks on IPsec, when configured according to the 3GPP specification and other configurations. The relevant IPsec standards are IETF RFC 2401–2412 [2-13]. The Ericsson CPP IPsec implementation to be used in Ericsson Ethernet Interface board will be referred as “Ericsson IPsec implementation” in this document. This thesis will cover only IP version 4 (IPv4), since the testing environment and the initial Ericsson IPsec implementation is IPv4. At this stage of the Ericsson IPsec implementation, no automatic key management protocol exists, e.g. IKE is **not** implemented. Therefore, this thesis will not cover the key management protocol and only manual keying is used, but **any actual Ericsson IPsec implementation will use IKE or other key management protocol**. To demonstrate the theoretical ideas for detecting attacks, a prototype without a graphical interface will be developed in C programming language.

1.2 Ericsson Platforms

Ericsson uses a layered network approach in order to structure a telecommunication system’s functionalities. The layers are: content and user application, communications control, and connectivity (see Figure 1). The layered system approach has evolved from the Ericsson’s earlier successful AXE telephone exchange system. Two new platforms extend the AXE platform [14]: Telecom Server Platform (TSP) and the Ericsson Connectivity Packet Platform (CPP), previously called Cello Packet Platform. These new platforms extend the existing AXE platform to handle both circuit switching and packet data transport.

The Ericsson AXE 810 platform supports Global System for Mobile communications (GSM) and Universal Mobile Telecommunications System (UMTS) networks. The AXE platform has many built-in functions, specifically it supports both a Mobile Switching Center (MSC) and Base Station Controller (BSC). Most Ericsson platforms are built with commercially available standardized components, interfaces, buses, and software.

TSP was introduced in order to support the second generation of mobile phone standards (2G). It handles content and user applications, and offers several network operator services in the same physical structure. TSP is able to offer network operator services, i.e. Home Location Register (HLR); Media Gateway Control Function (MGCF); Authentication, Authorization and Accounting (AAA) [15].

CPP [16] was introduced in order to support the third generation of mobile phone standards (3G) building upon GSM. CPP supports both Asynchronous Transfer Mode (ATM) and IP traffic. Physically CPP employs several chassis containing different kinds of circuit boards, processor boards, switch boards, interface boards, echo cancellers, and transcoders. CPP products are currently on the market and used to provide a variety of telecommunication services, such as RBS and RNC for Wideband Code Division Multiple Access (WCDMA) networks and Radio Access and Packet Data Serving Nodes (PDSNs) for Code Division Multiple Access 2000 (CDMA2000) networks.

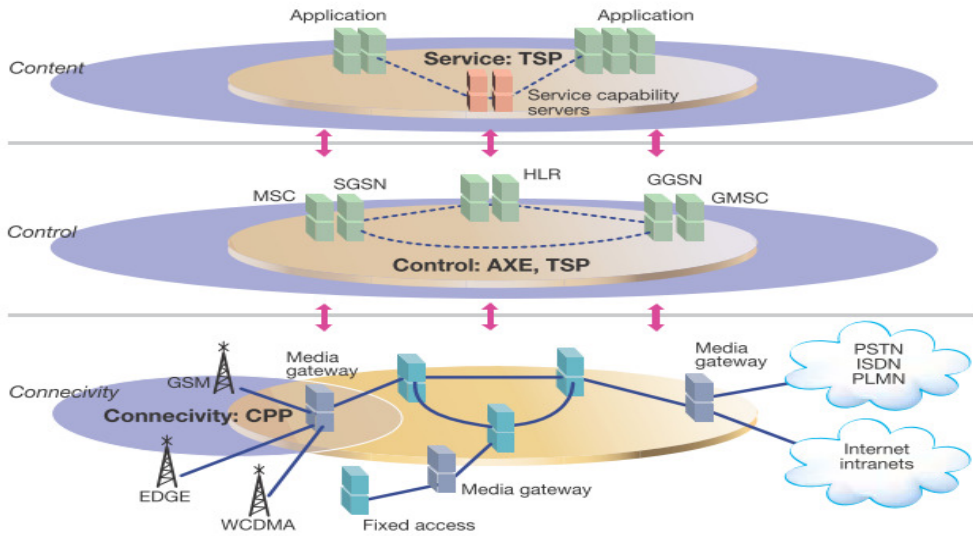


Figure 1: Ericsson Layered Platform Structure [14] ¹

1.3 Ericsson Ethernet Interface board

The Ericsson Ethernet Interface board is an Ethernet-to-ATM (Asynchronous Transfer Mode) backplane converter. The board can be used in Radio Base Stations or in a Radio Network Controllers with an LSI Corporation APP300 family (formerly Agere) network processor.

¹ This figure appears here with the permission of CGA Information AB.

The main modules of the board are:

Opto/Line Interface Unit (LIU) module	Optical and electrical Gigabit interfaces (1000BASE-X and 1000BASE-T)
Ethernet switch module	Ethernet switch (operates at the data link layer, layer 2 in the Open System Interface (OSI) model). In RBS nodes the switch is used for switching between the site LAN (temperature sensors and site equipment) and Operation & Maintenance (O&M) terminals and mainly as connection point for the IP/Ethernet backbone
Network processor module	Network Processor Unit (NPU), that allows IP termination and internetworking to ATM, and the IPsec block, to provide IPsec functionality
Device Board Module	Main Processor and memories. The main processor belongs to PowerPC family and is referred in Ericsson documentation as “Board processor”. It manages the traffic on the board, which includes board processing support, switch function monitoring and configuration, NPU monitoring & configuration, and NPU error & special packets (also called exception packet) processing

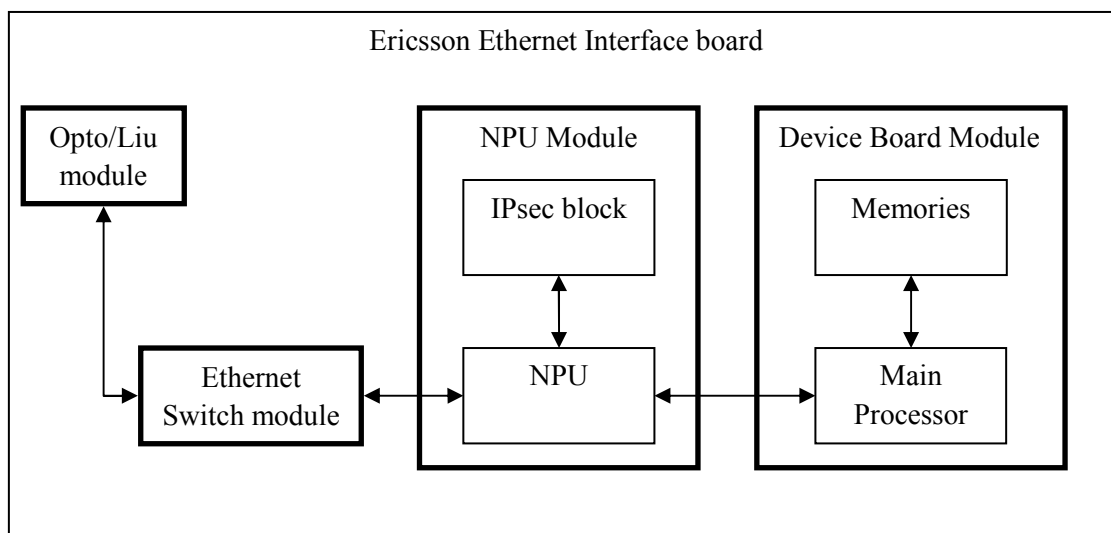


Figure 2: Main modules of the Ericsson Ethernet Interface board

1.3.1 Network Processor

Due to the enormous demand for bandwidth and computationally intensive applications in telecommunication environments, network processors have a crucial role in assuring high-speed data processing. For a good review on network processors see [18].

Network processors are used in an RBS or RNC to provide IP to ATM interworking, when this is necessary. In a network processor [19], time-critical processes, i.e. forwarding, shaping, etc., are executed in the data path/dataplane, also called the Wire-Speed Path since it receives the packets that need minimal processing; hence, these packets should be processed at the speed of the incoming packets on the relevant line interface. While management duties, i.e. error processing, configuration, aggregating and reporting of statistics, etc., are executed in the control path/controlplane, also called the Slow-Speed Path since it receives only packets that need unusual, less time critical, or complex processing. This division into fast path and slow path assumes that there is a division between packets which can be processed using the fast path and all other packets and **that the majority of packets do not need slow path processing.**

The network processor (NP) used on the Ericsson Ethernet Interface board is from the LSI Corporation Advanced PayloadPlus APP300 family of software-programmable network processors. The processor offers a fully integrated, single-chip solution with data-path and control-path functions offering a bandwidth in the range 400 Mbits/s up to 2 Gbits/s [20]. Each member of the APP300 family provides classification for processing protocols as well as policing/metering and statistics functions to enable flexible billing and accounting metrics for both ATM cells and IP/Ethernet packets. The APP300 uses SDRAM memories and standards interfaces to provide connectivity to the physical layer or backplane devices.

The APP300 Network processor series uses the Functional Programming language (FPL) [21] [22], to classify the incoming data, and the C for Network Processor (C-NP) language [23], to perform Traffic Management functions on the data processed by the classifier, to execute checks (policing), and collect statistics.

The main blocks of the APP300 are (see Figure 3):

Physical interfaces	data is received in the input interfaces and delivered to the output interfaces
Classification and Policing	incoming packets are stored in queues, pattern matching and classification is performed, and finally the packets are reassembled; during the process statistics are collected
Traffic management	enforces discard-policies, shapes the Quality of Service (QoS) and Class of Service (CoS), and performs any necessary packet modifications

The Agere Payload Plus Network [24] processor series is produced by LSI (following the merger of Agere corporation into LSI).

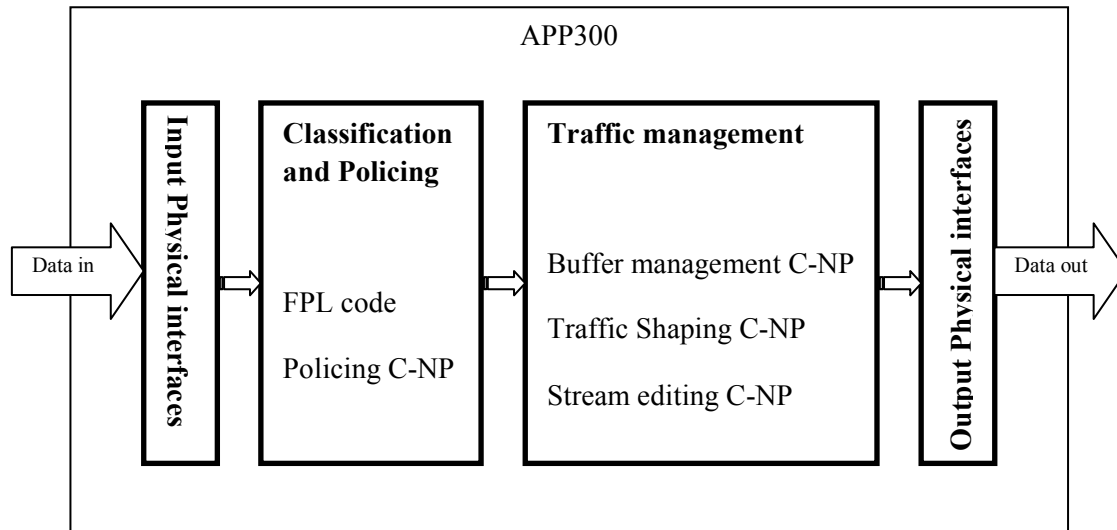


Figure 3: Main blocks of the APP300 Network processor series

1.3.2 IPsec processing in Ericsson Ethernet Interface board

The NP sends IPsec processing information to the IPsec block via SPI-3 by appending an extra header to the packet to be processed, called the Security Protocol Processor (SPP) transform internal header. Different headers can be added to the packet according to the desired IPsec functionality, inbound/outbound direction, and to request AH or ESP processing. The format of the first 3 words (32 bits per word) in each header is the same for all headers, but additional information can be added after the first 3 words.

During IPsec processing, the IPsec block removes the previously added SPP header and adds a two word SPP result header to the packet. This header contains error codes generated during processing (if any) and the (original) Opaque Software Tag from the incoming SPP header, see Figure 4.

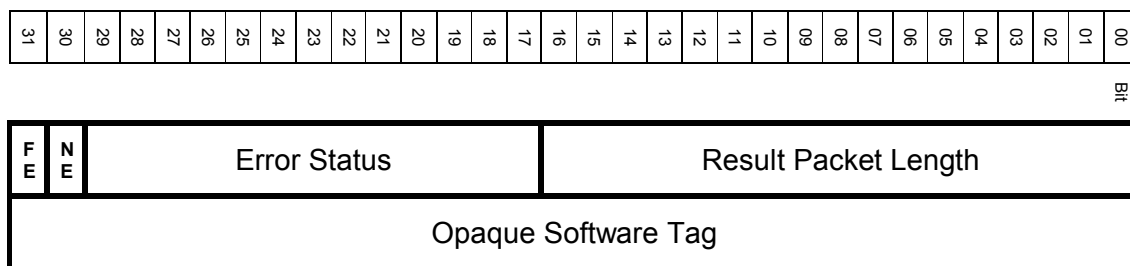


Figure 4: SPP Result Header

Fatal Error (FE)	Indicates if a fatal error has occurred. (Error codes are explained later in this section)
Non Fatal Error (NE)	Indicates if a non fatal error has occurred
Error Status	If FE or NE are set, this field indicates the type of error, otherwise it should contains zeros
Result Packet Length	Specifies the length of the packet processed
Opaque Software Tag	The 32 bit field specified in the SPP transform internal header passing unmodified

During processing, two types of errors might occur: fatal or non fatal errors. It is possible to have more than one error code per packet. Fatal error occurs only if the input token or the SA context record is incorrect, therefore they are due to programming errors. Note that a modified IPsec packet should not generate a fatal error, but only one or more non-fatal errors, see Table 2. The upper two bits of the SPP result header indicate whether an error has occurred, see Figure 4 and Table 1. Once a fatal error occurs, the packet must be discarded. If one or more errors occur, the Error Status field is set, see Figure 4. In this thesis we are only concerned with the non fatal errors that might be generated by (maliciously) modified IPsec packets.

Table 1: The two most significant bits in the SPP result header indicating type of IPsec processing error

Bit set	Type of Error
00	No error
01	Non fatal error
10	Fatal Error
11	Fatal Error and Non Fatal Error

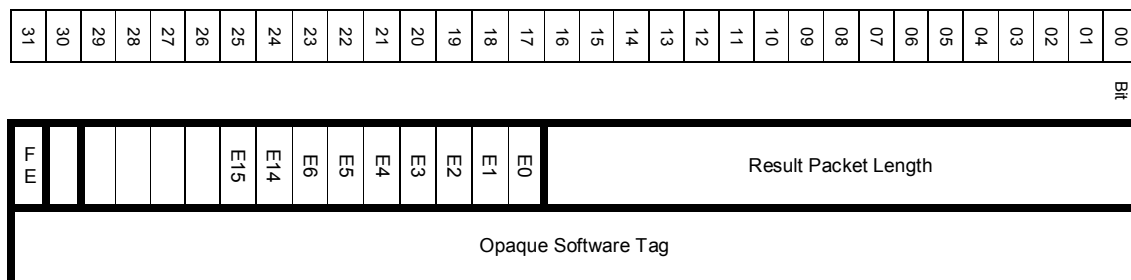


Figure 5: Position of Fatal Errors bits in SPP Result Header

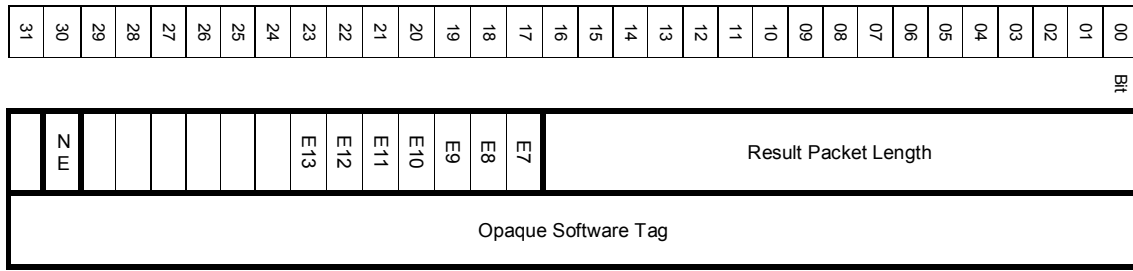


Figure 6: Position of Non Fatal Errors bits in SPP Result Header

Table 2: IPsec block Fatal Error Codes

Error ID	Description
E0	Packet Length error: token instructions vs. input fetch
E1	Token Error, unknown token instruction
E2	Token contains too much bypass data
E3	Crypto block size error(ECB, CBC) / Counter Overflow(CTR)
E4	Basic Hash: block size error
E5	Invalid algorithm/command/mode/combination
E6	Using algorithm which is prohibited by the virtue of SPP Core register 4
E14	Time-out Error (i.e. the SPP core hung and timed out)
E15	Output DMA Error

Table 3: IPsec block Non-Fatal Error Codes

Error ID	Description
E7	Basic Hash: hash input overflow
E8	TTL / HOP-limit underflow
E9	Authentication failed (Inbound)
E10	Sequence Number check failed / rollover
E11	SPI Check Failed
E12	Checksum Incorrect
E13	Pad Verification failed

1.4 Intrusion Detection

A definition of intrusion detection is given in U.S. Department of Commerce, National Institute for Science and Technology's (NIST) Special Publication on Intrusion Detection Systems (IDS) [26]: *"Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices"*. These incidents could be of malicious nature or caused by inexpert use of the system.

There could be several reasons to use Intrusion Detection Systems, depending on the aim of the system [27]:

- To monitor a system in order to track attempts against the security of the system
- To detect if a system is unprotected against attacks not prevented by other security measures
- To detect if an user is probing the system (retrieving information prior to perform an attack)
- To retrieve feedback concerning the security of the system
- To provide the history of a successful attack, in order to understand the cause and to restore the system

Telecommunication networks are a very important part of the modern world's infrastructure, therefore it is crucial to detect any "strange activities" on the network. Activities of interest could be an attack against a network (detection when the attack has already been performed), an attempt to attack the network (detection during the attack is being performing), or retrieval of information prior to performing an attack (also called *probing*). Furthermore, an Intrusion Detection System must be able to reveal possible vulnerabilities of the system by analyzing, tracking, and alerting the network operator.

The main parts of an IDS are:

Information Sources The sources of information used to determine whether an intrusion has taken place. In our work, the information source is the error codes generated, by the IPsec block during the IPsec processing and by the NP during SA lookup or Security Policy lookup

Analysis This part of the intrusion detection systems analyses the data retrieved from the information sources and decides if there is evidence of an attack against the system. The three main approaches to analyze events to detect attacks are:

Signature-based detection Based on a known attack pattern (also called a *signature*). The major limitation of this approach is that it only detects known attacks

Anomaly detection Based on comparison of abnormal system activity against normal system activity observed during a period of time (also called *profile*). This could be effective in detecting previously unknown threats, but the approach could also produce many false positive (false detection alarms) depending on how much the current traffic differs from the profile

Stateful protocol analysis or deep packet inspection Based on comparison of predetermined profiles of accepted state transitions of a protocol against observed events. This technique could detect unexpected sequences of commands in the execution of a protocol, but it cannot detect many forms of attacks, e.g., denial of service attacks, that execute the correct transitions in the protocol's states

In our work, we focus on the *Analysis* sub-system of an IDS. This sub-system is designed to run external to the network processor board and to manage several Ericsson Ethernet Interface board.

Response The set of actions that the system takes once it detects intrusions. These actions are basically divided into *active* response, i.e. automated intervention, and *passive* response, i.e., reporting the events. In our project, the *Analyzer* is designed to simply alert the operator in case of detected intrusions. Thus our solution will not include implementing any automatic responses to the detection of a potential attack also known as Intrusion Prevention System (IPS)

The placement of an IDS is of importance, as these systems are limited when IPsec protocol is used. The possible scenarios are described below. If the IDS is placed before an IPsec gateway, see Figure 7, the IDS will not be able to analyze encrypted traffic and therefore will not be able to detect attacks.

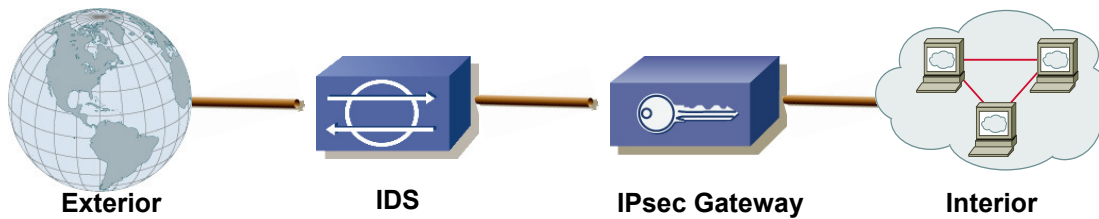


Figure 7: IDS placed before an IPsec gateway

Also when placing an IDS behind an IPsec gateway, as in Figure 8, the IDS will not be able to detect IPsec specific attacks, i.e. replayed packets, incorrect encrypted packet, DoS attacks, etc., since these packets are discarded before they are forwarded for IP processing, hence the traffic data is never forwarded to the IDS for analysis.

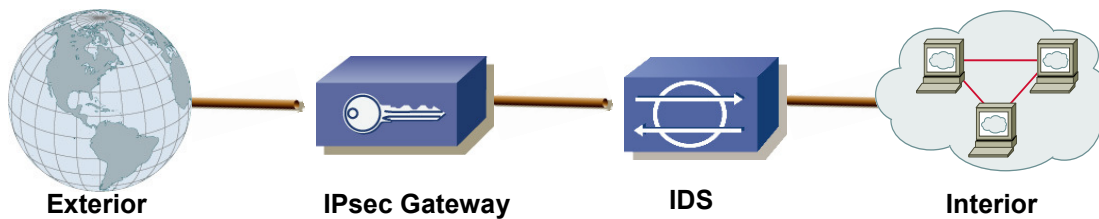


Figure 8: IDS placed after an IPsec gateway

In case of IPsec traffic, the best location for the IDS is incorporating it into the IPsec gateway, see Figure 9, in order to learn of processing failures: if any processing errors occur, the IDS will still be able to retrieve and analyze the data.

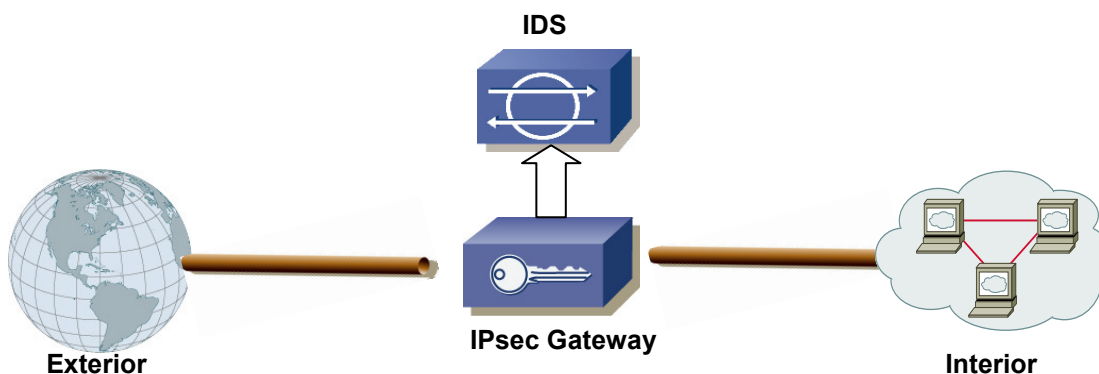


Figure 9: IDS placed in direct connection with IPsec gateway

An IDS that is able to detect attacks against IPsec gateways is therefore an interesting research topic. An IDS could have several outcomes as shown by Figure 10. The outcomes are:

- False Positive** The IDS alerts when no attack is occurring
- True Positive** The IDS alerts when an attack is occurring
- False Negative** The IDS does not alert when an attack is occurring
- True Negative** The IDS does not alert when no attack is occurring

		Attack	
		True	False
Alarm	True	True Positive	False Positive
	False	False Negative	True Negative

Figure 10: A matrix illustrating different IDS outcomes

The ultimate IDS should generate as few false positives and false negatives as possible. False positive errors usually require an operator’s attention to solve the situation and a high number of these alerts might lead to the operator to ignore future alerts. False negatives on the other side are the situations labeled as normal by the IDS, whereas an attack is occurring and the IDS does not fulfill its expected function.

2 BACKGROUND

2.1 The Internet Protocol

The Internet Protocol (IP) operates at the network layer in the Open System Interface (OSI) model and is a best effort protocol, consequently, it is unreliable and connectionless. There are currently two versions of the protocol: IP version 4 (IPv4) [28] and IP version 6 (IPv6) [29]. One of the differences between the two versions is the number of IP addresses: just over 4 billion of addresses for IPv4 (address size of 32-bit), over 16 billion-billion of addresses for IPv6 (address size of 128-bit).

At the IP layer, packets are called *datagrams*. A datagram is composed of a header part and a data part, the latter is also called an IP payload. In IPv4 the header can be 20 to 60 bytes long and contains the information for routing and delivering the packet. The data part can be 0 to 65515 bytes long. The maximum size of IP datagram is 65535 bytes, but the data link protocol imposes a limit to this size. For example, Ethernet frames can have maximum payload of 1500 bytes, therefore IP datagram over Ethernet cannot be longer than 1500 bytes. The limit on the maximum IP datagram size imposed by the data link protocol is called Maximum Transmission Unit (MTU).

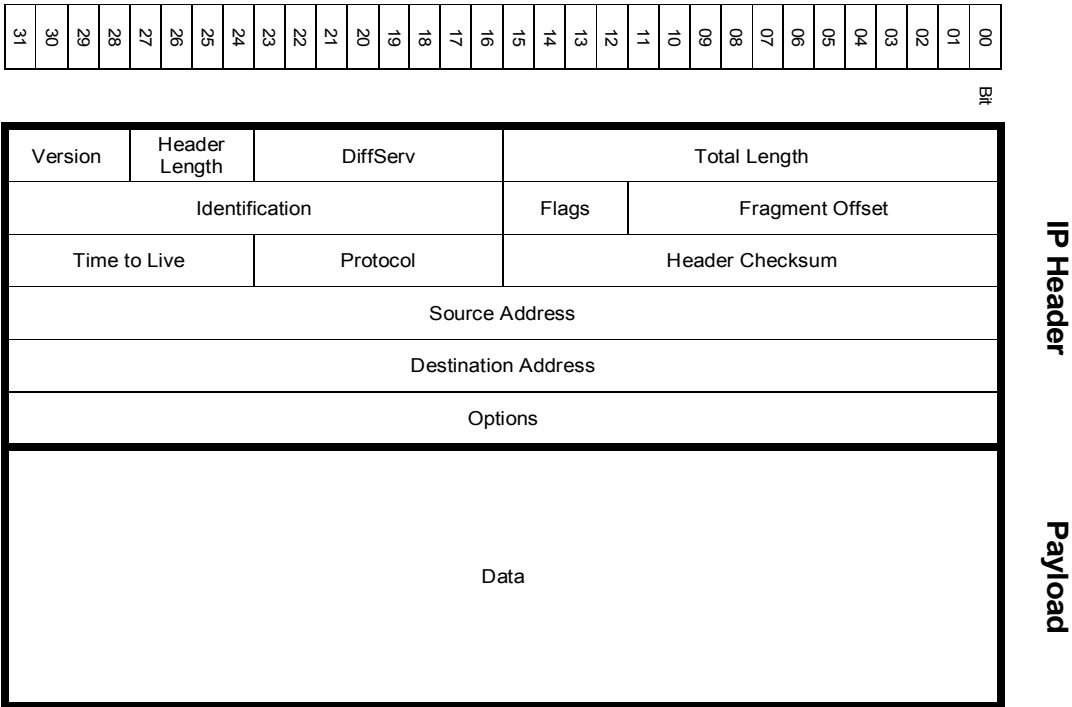


Figure 11: Format of IPv4 Datagram

Description of the header fields:

Version (VER)	This field contains the version number of the IP protocol. The version can be 4 (IPv4) or 6 (IPv6). In this thesis we are only concerned with IP version 4 (see Section 1.1)
Header Length (HLEN)	This 4-bit field defines the total length of the datagram header in 4 byte words and, consequently, specifies the offset to the data. The value of this field depends on the variable length of the header: if the Options field is not used, the length of the header is 20 bytes (the value 5×4 bytes = 20 bytes), otherwise the length could be up to 60 bytes (the value 15×4 bytes = 60 bytes)
Differentiated services (DS)	Previously called Type of Service [2] specifies a preference for how the datagram has to be handled (delay, throughput, reliability). Recently [30], this field has been redefined as <i>Differentiated Services</i> , due to the new technologies that require real-time data streaming and different classes of service
Total length	This field specifies the total size of the datagram in bytes (header plus data). The total size can be the minimum of 20 bytes (20 bytes header plus no data) and up to 65535 bytes. The length field is 16 bits long
Identification	This field is used during datagram fragmentation to identify the fragments of the original datagram
Flags	This field is used to control fragmentation (the Don't Fragment bit) or to indicate if there are more fragments (the More Fragments bit), the third bit is reserved and should be set to zero
Fragment offset	This field is used in fragmentation and specifies the offset of a fragment from the beginning of original IP datagram.
Time to live (TTL)	This field specifies the lifetime of a packet. Originally designed as the time in seconds, today it expresses the maximum number of hops the packet can transit. The TTL value is decremented every time a datagram is processed by a router, and the datagram is discarded once the value is zero

Protocol	This field indicates the protocol used for the payload of the IP datagram. The protocol-number is assigned by Internet Assigned Numbers Authority (IANA) [31]
Header checksum	This field is used to check the header for transmission errors; the checksum of the header must be computed and compared to the stored value of this header at every router (Section 3.1 in [28]); when this check fails, the datagram is discarded
Source IP address	This field contains the source IP address. However, if Network Address Translation (NAT) is used, the original source IP address is translated to a global network IP address associated with the NAT device or vice versa
Destination IP address	This field contains the destination IP address, but if NAT is used, then a global IP address is used in place of the original destination IP address or vice versa; this global address will be associated with the destination NAT
Options	This field is a an optional field used for application-specific information and it affects the length of the HLEN field

At the network layer the main processing steps are routing, fragmentation, reassembly, and Address Resolution Protocol (ARP) [32].

2.2 IP Security

IP Security (IPsec) was designed by Internet Engineering Task Force (IETF) to provide a security framework at the network layer, i.e., layer 3 of the OSI model. IPsec ensures private communication over a public IP network [2] and is commonly used to implement Virtual Private Networks (VPN) [33].

Depending on its implementation and configuration, IPsec could provide the following types of security services:

Confidentiality	IPsec prevents the disclosure of transmitted data to unauthorized parties through encryption. The data is encrypted using cryptographic algorithms and the receiver must have a key to decrypt the data
-----------------	---

Connectionless Integrity	IPsec detects the modification of transmitted data, without regard to the ordering of the datagram in a stream of traffic. The integrity is assured through a Hashed Message Authentication Code (HMAC), which is the output of a cryptographic hash function of the data
Data origin authentication	IPsec guarantees the origin of the data through cryptographic processing
Anti-Replay	IPsec is able to detect and reject replayed packets sent by an attacker
Access control	IPsec is based on policies that are applied to classify and filter the traffic. The policies can regulate the user's behavior or shape the network traffic

IPsec consists of the several components [34] [35]:

- **Security associations (SA)**
- **Security Policy Database (SPD)**
- **Security Association Database (SAD)**
- **Two Security protocols:** Authentication Header (AH) and Encapsulated Security Payload (ESP)
- **Two Modes:** Transport mode and Tunnel mode
- **Cryptographic algorithms:** used to provide authentication (i.e. MD5 and SHA-1) and encryption (i.e. DES, TDES, Blowfish and AES)
- **Key management:** Internet Key Exchange protocol (IKE) or manual keying

2.2.1 Security Association

Since the IP protocol is unreliable and connectionless, Security Associations (SA) are used to create a secure and protected connection between two hosts. An SA is a unidirectional association between a pair of hosts, therefore, for two-way IPsec communication, two SAs are needed, one for each direction (i.e. inbound and outbound traffic). Either an automatic key management protocol (IKE) is used for SA negotiation or it is set manually. Every SA is **uniquely identified** by three basic elements:

- **Security Parameter Index (SPI):** is an arbitrary 32-bit value selected by the destination to identify the SA's entry in the database. The same SPI can be assigned to a different SA (for a given destination address), but only if a different security protocol is used (thus there is an SPI for each of AH and ESP)
- **Security protocol:** AH or ESP

- **Destination IP address:** unicast address, IP broadcast, or multicast group address. There can be **many** SAs associated with a given destination address.

The SA stores also information about IPsec mode, cryptographic algorithms, cryptographic keys, and lifetime of the keys agreed by both peers for the communication (see the example in Table 4).

Table 4: Sample of Security Association

Source Address	192.168.66.10
Destination Address	192.168.70.10
Security Parameter Index	1000
Mode	tunnel
Security Protocol	ESP
ESP algorithm	AES-CBC 128 bits
ESP algorithm key	“123456789....”

2.2.1.1 Security Policy Database

The Security Policy Database (SPD) [2] consists of a list of policy entries. This database is consulted during the processing of all traffic, IPsec and non-IPsec. Each entry in the SPD must specify one or more selectors (depending on the granularity of the actions to be taken) and a processing action. The selector is a set of IP or upper layer protocol header field values, used by the SPD to map traffic to a policy. The selectors can include Destination IP address, Source IP address, Transport Layer Protocol, System Name, and User ID ². The SPD three possible processing actions are:

- | | |
|---------|---|
| Discard | the traffic is not allowed to pass and is discarded |
| Bypass | the traffic is allowed to pass further without any IPsec protection |
| Protect | the traffic requires IPsec protection |

The SPD processing scheme is different for inbound and outbound traffic. During both processing phases, an audit log should be utilized.

² Not all selectors might be available

2.2.1.1.1 IPsec Inbound Processing

Inbound processing [2] is performed on the incoming IPsec-protected packets. If the received packet was fragmented, the reassembly of the packet is performed prior to the AH or ESP processing. After the packet is reassembled, the `Fragment offset` field is set to zero and the `Flags` field is reset.

When a packet is received, the following steps are executed:

1. The SPI, the packet's destination address (outer IP header), and the Security Protocol are used to look up one or several SAs in the SAD
2. If a SA is found. The following steps are taken:
 - a. If anti-replay protection is enabled, the Sequence Number is validated
 - b. If integrity protection is enabled, the hash of the authenticated data must be verified by computing the Integrity Check Value and checking if it matches the value in the HMAC field see Figure 12
 - c. If confidentiality protection is enabled, the packet is decrypted and the original IP datagram is de-encapsulated or (re-)constructed (depending upon the mode used)
 - d. The inbound SPD is consulted to determine what policies and actions should be applied
 - e. A check to see whether the required IPsec processing has been applied according to the selected policies is done
 - f. Forward the packet to IPsec outbound processing or send the datagram further for IP processing
3. If no SA is found, the packet is discarded and an error should be logged

2.2.1.1.2 IPsec Outbound Processing

Outbound processing [2] is applied to packets prior to transmission. If a packet has to be fragmented, IP fragmentation occurs after IPsec processing. The SPD is consulted during the processing of IPsec and non-IPsec traffic, to determine what kind of action is to be taken. The outbound policies in the SPD are matched against the packet's selector fields to locate the first of the appropriate policies. This can result in a decision to discard the packet, bypass IPsec processing, or protecting the packet with IPsec. In the last case a number of further steps are taken. These are:

1. The policy will point to zero or more SAs in the SAD
2. If one or more SAs exist, match the packet's selector fields against those in the existing SAs. Otherwise, if no SAs were found or none match, the packet is dropped. Note that while normally an IPsec implementation would create a new SA using IKE, since currently we only support manually established keys, the absence of a relevant SA is an indication of either an error or an attack and hence should be reported to the IDS
3. IPsec header is added to the packet (more formally: the existing packet is encapsulated in a new IP packet (Tunnel mode))

4. The selected SA is used for IPsec processing: if confidentiality protection is required, then encryption is performed. Afterwards, if integrity protection is required, a keyed hash value is calculated and added to the new packet

2.2.1.2 Security Association Database

The Security Association Database (SAD) [2] contains the parameters associated with each active security association. Each entry in the SAD corresponds to a security association and it must contain the value or values negotiated when the security association was created. The entries in the SAD are ordered because a security policy may require more than one SA to be applied to a specific set of traffic. For example, the ESP SA might be applied first and then the AH SA (or vice-versa); each order has a particular semantics. The way that the SAD is consulted during inbound processing is different from how it is consulted for outbound processing, see Sections 2.2.1.1.1 and 2.2.1.1.2.

2.2.2 Security modes

IPsec has two modes: transport mode and tunnel mode. The transport mode is usually used between two hosts to protect the end-to-end communication. This mode could also be used between a host and a security gateway, if the security gateway acts as a host.

Tunnel mode is used mainly for secure IP tunneling where one or both the ends of the SA are security gateways that forward the traffic. This IPsec mode is widely used to create VPNs. This mode could also be used for host to host communication, and in this case the source and destination addresses would be the same in the inner and outer IP header. In our thesis project we work with ESP tunnel mode.

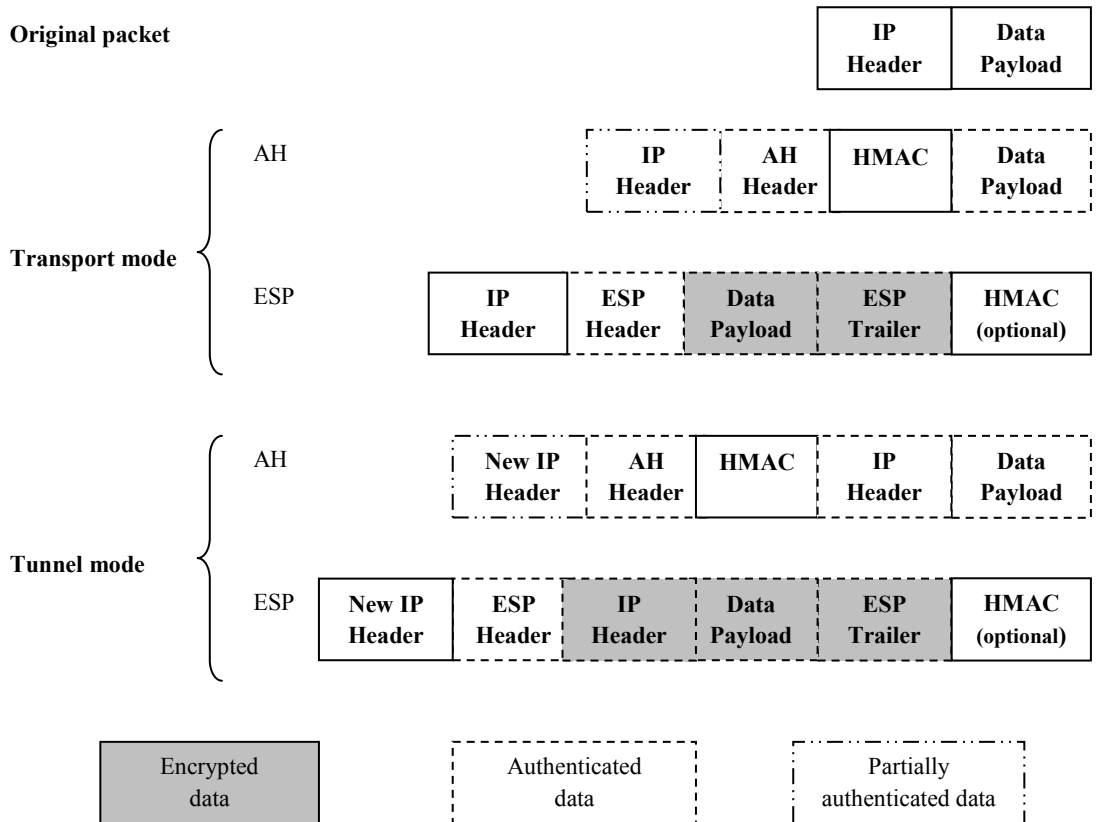


Figure 12: Format of an IP datagram showing transport mode and tunnel mode in comparison to the original packet

2.2.3 Security protocols

2.2.3.1 Authentication Header

The Authentication Header protocol (AH) [3] can provide data origin authentication, connectionless integrity, and optionally anti-replay protection for IP traffic. Authentication and integrity checks are performed by applying a hash-based algorithm (such as MD5 or SHA-1) to the packet, excluding the outer IP header's mutable fields: TOS/DS, Flags, Fragment Offset, TTL, and header checksum. Anti-replay is performed by including a unique sequence number in each packet. Usually a monotonically increasing counter is used for this purpose, with an initial value of zero.

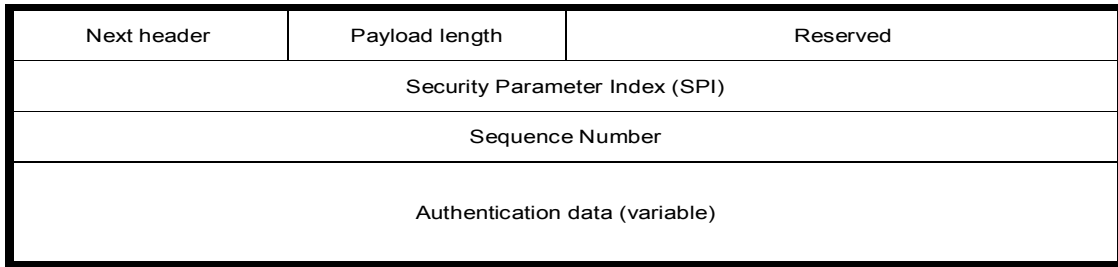
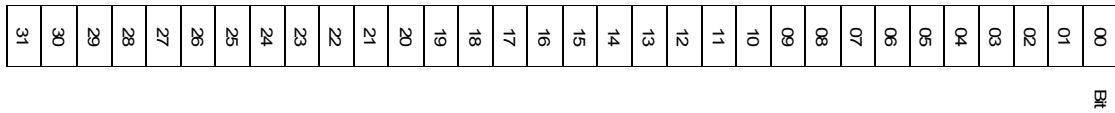


Figure 13: AH Header

Description of the fields:

Next Header	This field identifies the protocol of the data in the payload. It is the same as Protocol field value in IP header for transport mode and IP version value for tunnel mode
Payload Length	The size of the AH header expressed in 32-bit words
Reserved	Reserved for future use and must be set to zero
SPI	See Section 2.2.1, used to locate a SA in the SAD
Sequence Number	Used for anti-replay protection, a monotonically increasing counter value. It is mandatory for the sender to include the sequence number in a transmitted packet, but it is optional for the receiver to process it. The counters on both sides of transmission are initialized to zero when an SA is established and must be reset by establishing a new SA and thus a new key as soon as 2^{32} packets has been sent. In case of manual keying, anti-replay protection should not be used (according to Section 5 in [7]). If anti-replay is disabled, the sender still increments the counter value and it will start from zero after reaching the maximum value (according to Section 3.3.3 in [7])
Authentication Data	Contains the Integrity Check Value (ICV) of the packet, see Section 2.2.5, and must be multiple of 32-bits for IPv4. The recipient computes a hash over the received packet and compares the two hash values (i.e., calculated versus received): if they do not match the packet is discarded

2.2.3.1.1 AH in Transport Mode

AH in Transport mode contains the value 51 (AH) in the `Protocol` field, which indicates that there is an AH header, see Figure 14. The `Next header` field in the AH header specifies the protocol value of the packet actually encapsulated in the payload. In transport mode, this packet has to be the upper layer protocol, i.e. TCP, UDP, or STCP. During inbound processing, once all the IPsec fields are stripped off and the original IP packet is reconstructed, the `Protocol` field of the reconstructed IP packet will contain the value saved in `Next header` field in the AH header.

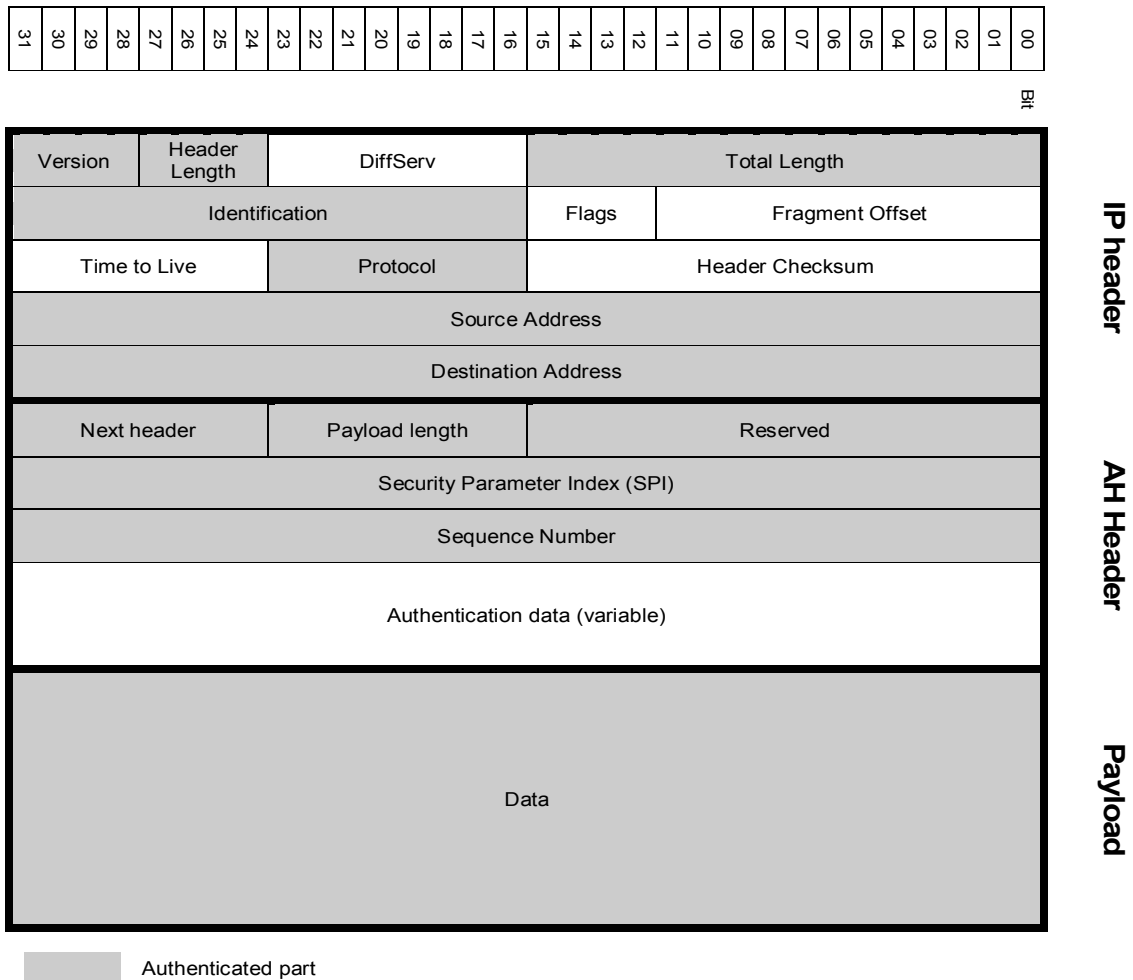


Figure 14: Format of IPv4 datagram protected with IPsec AH Transport mode

2.2.3.1.2 AH in Tunnel Mode

For AH in tunnel mode, the original IP datagram is appended after the AH header, and a new outer IP header is prepended (see Figure 15). The `Next Header` field contains the protocol value of the encapsulated packet, this is always 4 (IPv4) for tunnel mode. The `Source Address` and `Destination Address` fields might be different from the original source and destination address included in the original IP header (since they now represent the tunnel end-points). This creates a tunnel which can be used to implement a VPN.

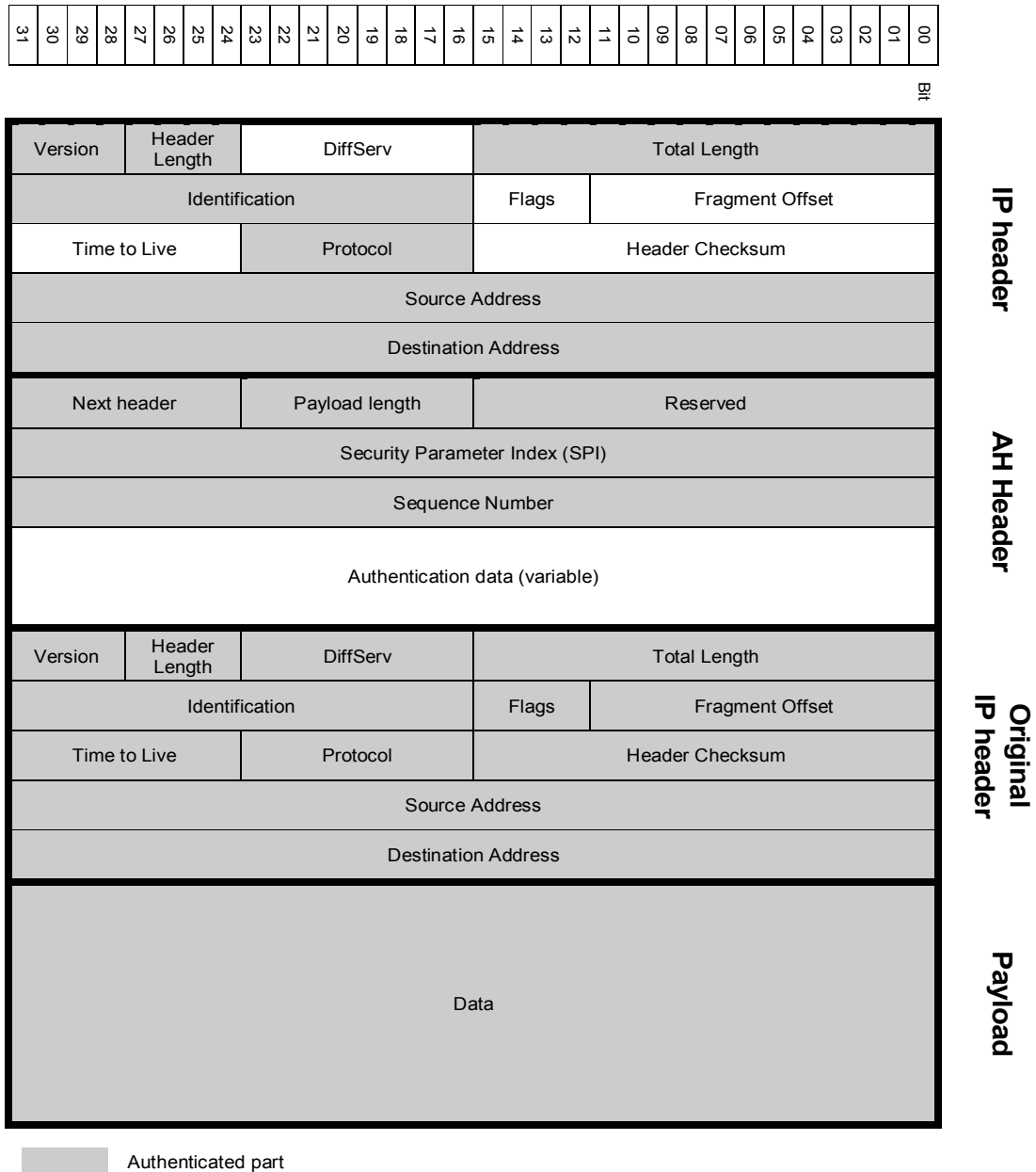


Figure 15: IPv4 Datagram Protected with IPsec AH Tunnel Mode

2.2.3.2 Encapsulated Security Payload

Encapsulated Security Payload (ESP) [7] provides confidentiality and optionally, data origin authentication, connectionless integrity protection, and anti-replay protection. If the encryption algorithm requires a random initialization vector (IV), the value is stored just before the protected payload (indicated as the Encrypted Payload in Figure 16). ESP includes a header and a trailer which surround the encrypted payload. The SPI, Sequence Number, and Next Header fields have the same function as in AH, however in the case of

ESP the Next Header is encrypted, therefore it is not possible to determine if transport or tunnel mode is used until the packet has been decrypted. In comparison with AH, the new fields are:

- **Padding:** this field enables use of a block-cipher for encryption by adding a variable amount of padding data, in order to make the encrypted data a multiple of the block size required for this block-cipher
- **Padding length:** this field specifies the length of the padding
- **Authentication data:** this field is optional and provides integrity only for the ESP header and encrypted payload, not for the full IP packet as is the case of AH

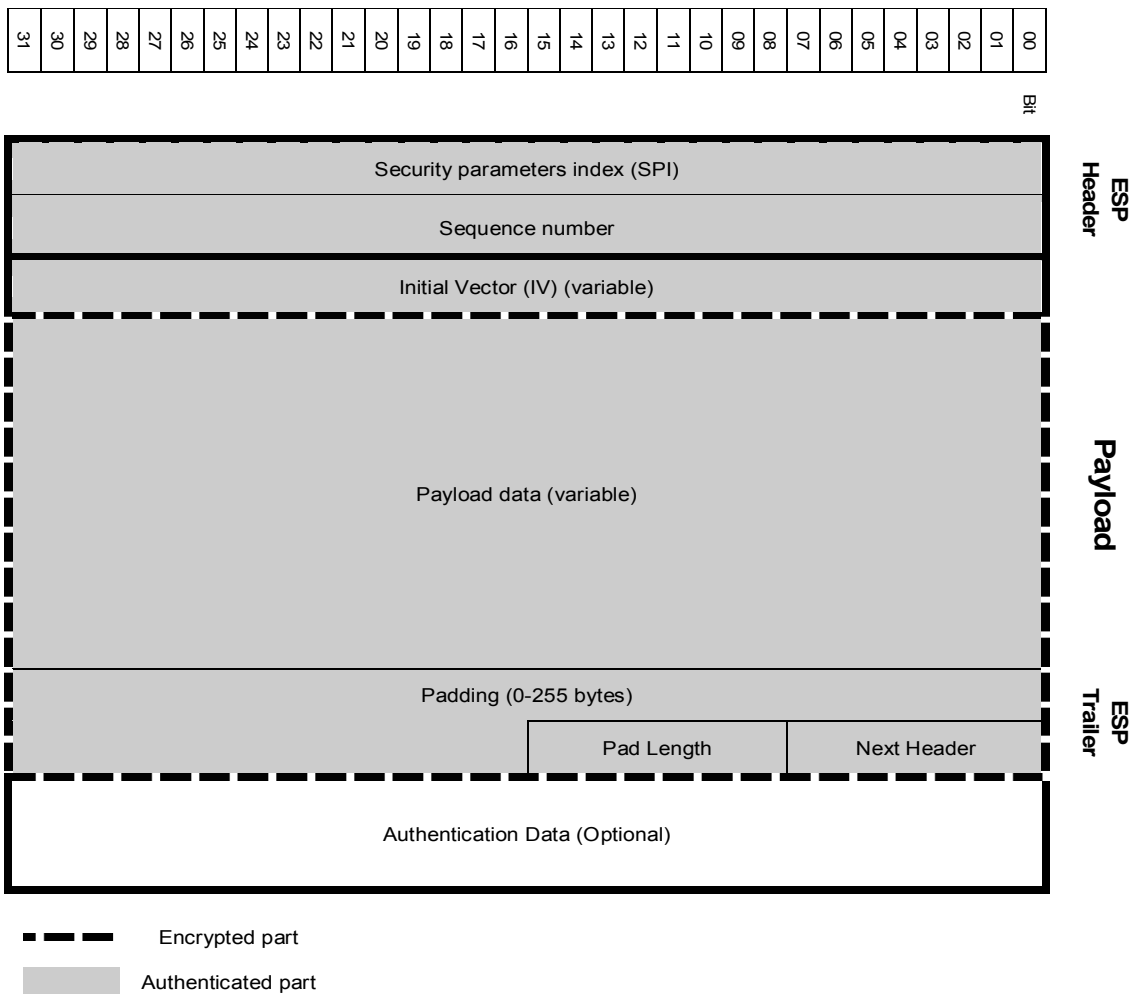


Figure 16: ESP packet

2.2.3.2.1 ESP in Transport Mode

For ESP in transport mode, security services are provided only for ESP fields and the higher layer protocol payload (i.e., TCP, UDP, and others) not for the IP header preceding the ESP header.

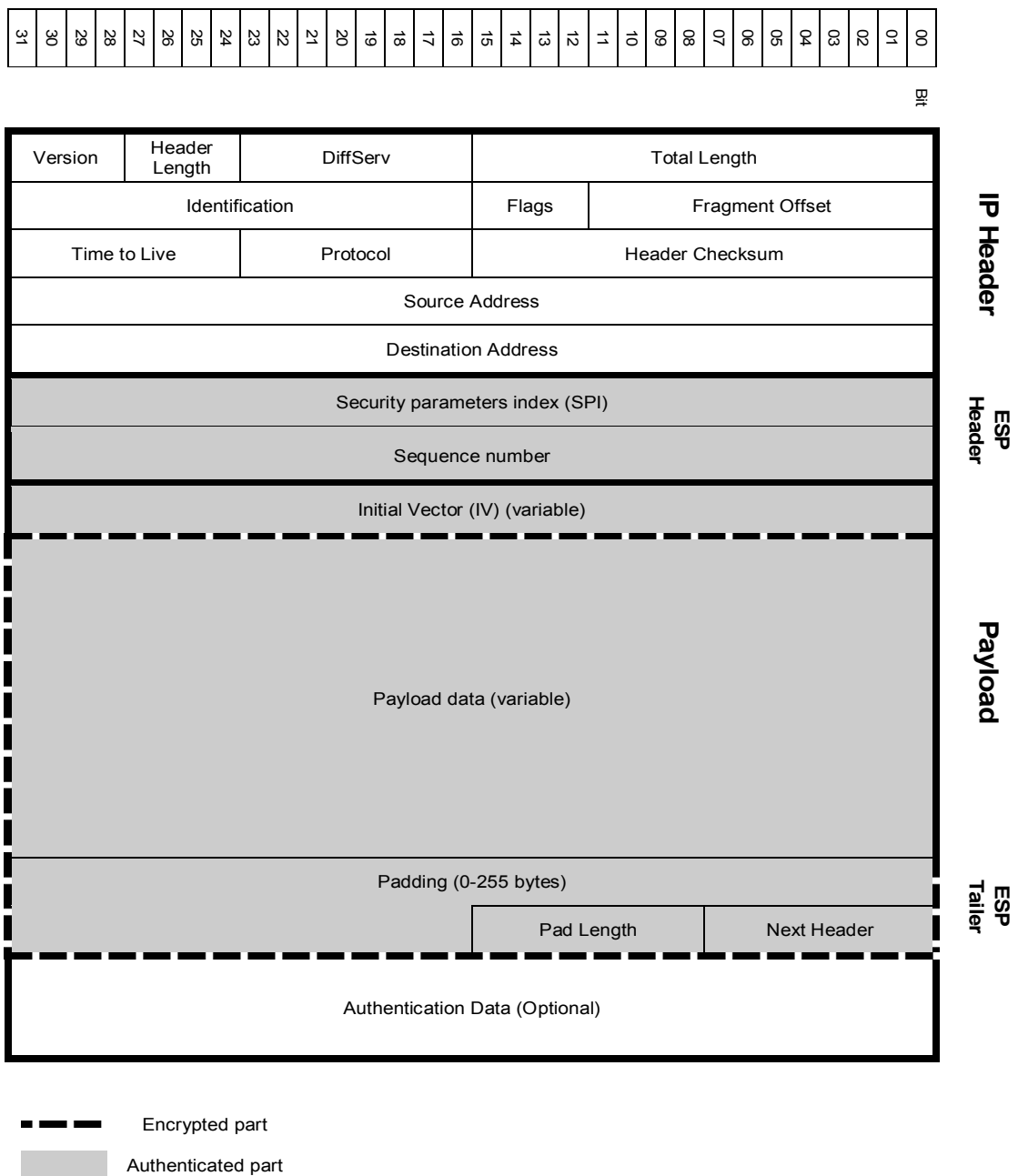


Figure 17: IPv4 Datagram Protected with IPsec ESP Transport Mode

2.2.3.2.2 ESP in Tunnel Mode

For ESP in Tunnel mode, the original IP datagram is encapsulated between the ESP header and the encrypted payload. The protection is applied only to the ESP header and original IP datagram. The `Source` and `Destination` fields of the new IP datagram could be different from the original source and destination address included in the original IP datagram (as these represent the end-points of the tunnel).

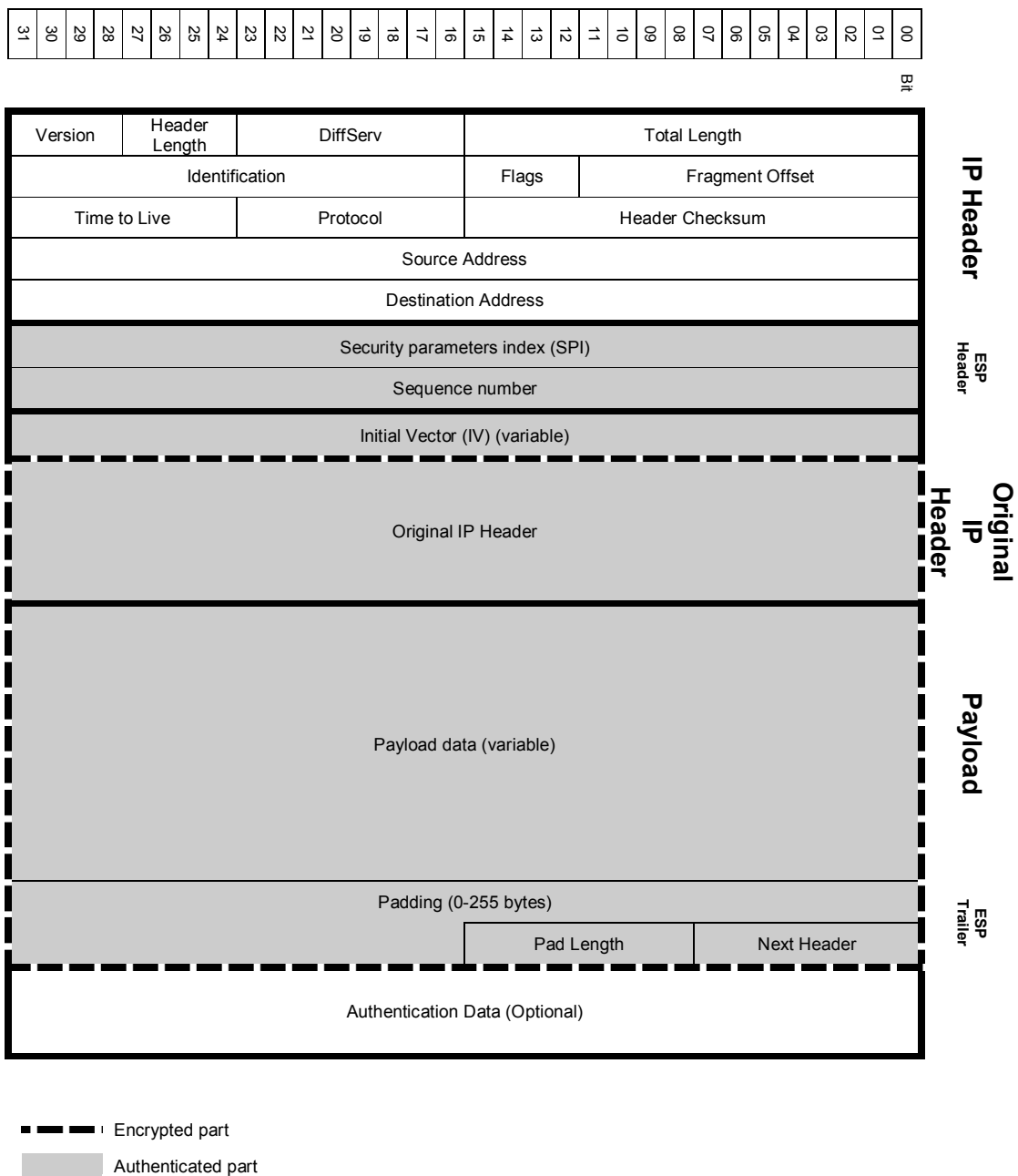


Figure 18: IPv4 Datagram Protected with IPsec ESP Tunnel Mode

2.2.4 Authentication algorithms

An Integrity Check Value (ICV) is calculated to assure the integrity of the packet. The mechanism to create the ICV is based on a secret key, shared between the two parties, and a hash algorithm such as MD5 and SHA-1. This mechanism is called a Hash Message Authentication Code (HMAC) [36]. HMAC can be used in both AH and ESP to assure integrity and data origin authentication. According to the 3GPP specification [1], the authentication algorithms that can be used are MD5 and SHA-1, although both of them have been

compromised. There are some authentication algorithms that haven't been compromised, e.g., SHA-224, SHA-256, SHA-384, SHA-512, but only MD5 and SHA-1 will be used in this project.

2.2.4.1 MD5

Message Digest 5 (MD5) algorithm [37] is a cryptographic hash function producing a 128-bit hash value. It was designed by Ronald Rivest in 1991 to replace MD4. Though some vulnerabilities have been found [38], MD5 is still a widely used hash function. The use of MD5 within AH and ESP for IPsec is specified in [4].

2.2.4.2 SHA-1

Secure Hash Algorithm (SHA-1) was designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) in FIPS PUB 180-1 and FIPS PUB 180-2 (the latest). SHA-1 is one of the five cryptographic hash functions named SHA and produces a message digest of 160-bit. The security of SHA-1 has been compromised by cryptography researchers [39]. SHA-1 was developed to be the successor to MD5. The use of SHA-1 within AH and ESP for IPsec is specified in [5].

2.2.5 Encryption algorithms

Encryption algorithms, also called ciphers, are used to encrypt data prior to transmission, transforming the content of the packet in order to assure confidentiality. Encryption algorithms operating on data blocks of fixed length are called block ciphers. To avoid the repetition of blocks produced by encrypting the same plaintext block, alternative encryption modes could be used or an Initialization Vector (IV) can be used to encrypt the first block of the data.

According to the 3GPP specification [1], Cipher Block Chaining (CBC) mode is used in ESP for 3G systems. This mode requires an IV to encrypt the first block and to randomize the process. The IV should be long enough that it will not be used twice in the encryption process with the same secret key. When manual keying is used, the secret key should be updated when all the possible IV values have been used. When IKE is used, a new SA will be negotiated once the Sequence number is close to an overflow. The use of CBC mode for encryption algorithm with IPsec ESP protocol is specified in [40].

According to Section 5.3.3 of the 3GPP specification [1], the encryption algorithms that must be supported are DES-CBC, TDES-CBC, and AES-CBC 128 bit.

2.2.5.1 DES

The Data Encryption Standard (DES) is a cryptographic algorithm selected by NIST as an official Federal Information Processing Standard (FIPS) [41] in 1976. DES can nowadays be easily broken since the 56-bit key length is too small [42], therefore it is no longer considered to be sufficiently cryptographic. The use of DES in CBC mode within ESP is specified in [6].

2.2.5.2 TDES

Triple DES (TDES) is a cipher derived from DES. It encrypts a plaintext block through DES three times and its key length could be 112 or 168 bit. The EEE mode is implemented by encrypting a block using DES three

times. The EDE mode is implemented by encrypting, decrypting and finally encrypting a block using DES. The use of TDES in CBC mode within ESP is specified in [43].

2.2.5.3 AES

The Advanced Encryption Standard (AES), also known as Rijndael, is a block cipher adopted as a Federal Information Processing Standard by NIST [44]. It has a variable key size of 128, 192, or 256 bits and a block size of 128 bits. The encryption process is done by iterating 12-14 encryption rounds, depending on the key size. The only successful attack against AES is side channel attacks: what is attacked is not the underlying cipher but the implementation of the cipher on the system, which leaks some data. Daniel Bernstein performed a cache timing attack using chosen cipher text and cache timing characteristics to calculate the correct key [45]. The use of AES within ESP is specified in [46] (for AES-CBC) and [47] (for AES-CTR).

2.2.6 Key Management

2.2.6.1 IKE

Internet Key Exchange (IKE) protocol [10] is an automated key management protocol used to set up a SA in the IPsec protocol suite. IKE is built upon the Oakley protocol, a key-agreement protocol based on Diffie-Hellman key exchange, allowing authenticated parties to set up a shared session secret, and to subsequently derive cryptographic keys. Mutual authentication of the communicating hosts is provided using public key techniques or a pre-shared key.

2.2.6.2 Manual Keying

Manual keying is the process of manually configuring keys and security associations on each host. Manual keying can be used in small static environments, but it is not advisable for network with many hosts, due to configuration and re-keying limitation. In our project manual keying is used because the Ericsson IPsec system is currently in development and no automatic key management protocol has been implemented yet.

2.2.7 Implementations and Architectures

IPsec can be implemented in a host or in a router. Host-implementation is the most flexible solution to provide end-to-end protection between a pair of devices in a network. However, IPsec need to be configured on each of them. Router-implementation (in this case the routers are also called a *Security Gateway*) is mainly used in tunnel mode to create a VPN, but can also be used in transport mode if the router is the final destination. The latter implementation is recommended if the network consists of many clients, to avoid IPsec configuration on each client, but leaving the connection between routers and local hosts unprotected by IPsec.

In our project a router-implementation using tunnel mode is used, as specified in Section 4.5 and Section 5.3.2 of the 3GPP specification [1]. Three different IPsec architectures are defined in RFC 2401 (The term “implementation” is used instead of “architecture” in the documentation):

- **Integrated architecture:** integrates IPsec protocols and capabilities directly into IP. This is the most elegant solution since no hardware or architectural layer has to be changed or reconfigured. IPv6 integrates IPsec as a mandatory part and the use is optional; while IPv4 requires making changes to the IP implementation on each device
- **Bump In The Stack (BITS) architecture:** IPsec is implemented as separate architectural layer between IP and the data link layer and adds security protection to datagrams created by the IP layer. This approach is appropriate for use with legacy systems and used for IPv4 hosts
- **Bump In The Wire (BITW) architecture:** an outboard crypto processor is used to provide IPsec services. This architecture allows non-IPsec compliant routers to provide IPsec services to the local hosts

Since IPsec is based on cryptographic computation, an architecture implementing IPsec has to supply the computing power for cryptographic algorithms. The current generation of network processors provides the computational capacity for IPsec, and a dedicated fast connection is used in order to offload the processing of memory expensive computation (look-ups for SA, encryption, and so on). In our project the BITW architecture is used since IPsec block is used for IPsec processing in the Ericsson Ethernet Interface board, see Section 1.3.

2.3 Related work

For commercial implementation of IPsec, the common way to warn users about a system's vulnerabilities is issuing a security advisory. Common Vulnerabilities and Exposures (CVE) [48] is a list of possible known security vulnerabilities and exposures, each of them identified by a standard name called *CVE Identifier*. Therefore, for CVE compliant IPsec implementation, the solution to remediate a problem can be searched for in the CVE databases. If an entry in the database matches the CVE Identifier, then the problem can be easily solved.

In case of proprietary implementations, an attack on the system must be detected in order to alert the operator, who can take manual actions, e.g., applying filters to discard attack traffic and neutralize the effects of these attacks.

Instead of taking manual action, a better solution would be to automate the process. VPNshield [49] is a prototype for protecting edge networks and detecting denial of service attacks, thus enabling continuous system operation. This product provides, as stated in [50], automated attack detection and a response mechanism to enable uninterrupted VPN service, automatic installation of filters to quickly eliminate the attack traffic. Currently, there is no version of this available as a commercial product.

In Mika Müller's master thesis, entitled "*Analysis tool for studying IP security Denial of Service resistance*" [51], DoS attacks against IPsec's IKE protocol are examined. The resistance of different IKE implementations were tested and evaluated. This thesis will likely be relevant to Ericsson and others who carry on the work begun in this thesis but it does not cover the IPsec protocol itself.

Ari Muittari's master thesis, entitled "*Internet Key Exchange (IKE) protocol vulnerability risks*", [52] examines vulnerabilities and possible ways to attack IKE version 1, reporting on experiments and the tools used to perform these attacks.

In Matti Järvinen's master thesis, entitled "*PKI Requirements for IPsec*" [53], the PKI and IKE protocol are investigated, to define the interface between IKE and PKI and enable a certificate-based authentication in IKE implementation.

"*Attacking the IPsec Standards in Encryption-only Configuration*" [54] show that some attacks against encryption-only IPsec configuration can be performed, even if the IPsec configurations reflects RFCs (2401-2411 or 4301-4309) and does proper padding checks. They tested their attacks on different open-source IPsec implementation and report the success of the attacks on the OpenSolaris implementation. They also point out the need for more precise prescriptions of how to handle security-sensitive issues, instead of just warning the IPsec manager on performing checks that, if badly implemented, could make the IPsec implementation vulnerable to old attacks (i.e. Bellare's attacks – see Sections 3.2 and 4.4.1).

The website of Arnold K. L. Yau, [55], a doctoral student of prof. Kenneth G. Paterson at University of London. This website contains links to both theoretical papers and attacks on IPsec, as well as articles on several different IPsec implementations.

Henrik Dikvall's master thesis "*IPsec in hardware*", [56], discusses how the IPsec protocol can be implemented efficiently in hardware, and in particular in a network switch, to make it acts as a security gateway. The thesis is motivated by the need to run IPsec on application specific hardware at "wire-speed", instead of as software on common CPU, because of the demanding computations needed.

3 CRITICISM AGAINST AND WEAKNESSES IN IPSEC

The IPsec protocol [2] has been criticized for being too complex. It tries to support many different situations with different options, which makes it hard to analyze its security thoroughly. There are two different modes that can be applied to each of the two security protocols, leading to four possible situations: ESP tunnel mode, ESP transport mode, AH tunnel mode, and AH transport mode. Add to that the possibility to choose from a number of encryption and authentication algorithms in different combinations. ESP mode must also support no encryption mode (NULL Encryption algorithm) to provide authentication and integrity without confidentiality and no authentication mode, which both could be used for debugging/testing.

This extensive variety of configurations makes IPsec hard to overview. Network administrators with limited knowledge in cryptography might find it difficult to configure an IPsec network in a secure way. Security experts have shown that certain security configurations make IPsec vulnerable and in some cases useless, as shown later in Section 4.

3.1 Ferguson and Schneier evaluation

Cryptography researchers Niels Ferguson and Bruce Schneier published a lengthy evaluation of IPsec [57] in 1999. The writers point out several weaknesses in the protocol and blame the complexity of the protocol on the way IPsec was developed, through a standardization committee. Too many options and too much flexibility was a side effect, often resulting in several ways of doing the same or similar things. At the end of the development process, nobody seems satisfied with the results. A better solution would be, according to the writers, to organize a contest similar to the one organized by NIST for the development of the AES algorithm.

IPsec documentation is also criticized for being hard to read and missing important sections in the RFCs, e.g., rationale and overview section. It is pointed out that the ISAKMP specification [9] contains errors, contradictions, and is missing essential explanations.

The writers question the need of having a separate *transport mode* and see it as a subset of the *tunnel mode*. Tunnel mode provides the same services as transport mode, and the writers recommend the *elimination* of the transport mode. The difference between the *ESP protocol* and *AH protocol* is also discussed. While the AH protocol authenticates more data than ESP, i.e. the outer IP header in tunnel mode, since ESP could authenticate the entire packet payload including the inner IP header in tunnel mode, this should be sufficient to convince the recipient that the packet was sent by someone in possession of the authentication key. There is no real explanation in the IPsec documentation, why the IP header should be authenticated, since IP is only used to transport the packet from one destination to another. Therefore, the writers recommend the *elimination* of the AH protocol.

The ESP specification [7] states that authentication is optional. This enables the configuration of encryption with no authentication, which makes IPsec vulnerable to very serious attacks as described in Section 4.4. The writers recommend that authentication should *always* be used, but that encryption should be optional. The Security Associations are also discussed. Recall that each SA is *unidirectional*, thus to be able to send data between two IP end-points, two SAs need to be negotiated. If encryption is provided by ESP and authentication by AH, then a total of *four* SAs must be negotiated. However, the need to be able to send data in one direction is unusual and a *bidirectional* SA would reduce the number of SAs required by the half and

reduces the SA negotiation setup exchange. Of course this latter point is irrelevant in the case of manual keying, since the SAs are statically configured.

Ferguson and Schneier [57] point out that IPsec ICMP processing is unclearly documented. Specifically, the Security Architecture for IP [2], Section 6, states that unauthenticated ICMP messages should be discarded since they could be used in a DoS attack. Since IPsec packets may traverse through routers not implementing IPsec and hence incapable of providing IPsec authentication, these ICMP messages would be discarded, and a critical function of IP would be lost. Local security policy determines if an IPsec implementation should reject or accept unauthenticated ICMP traffic. Section 6 in RFC 2401 [2] states the following “*Thus it MUST be possible to configure an IPsec implementation to accept or reject (router) ICMP traffic as per local security policy.*”. For instance, Microsoft’s Windows 2000 Server IPsec implementation allows the user to configure an IPsec filter list per policy [58].

3.2 Probable plaintext in IPsec

Steven M. Bellovin published an article about probable plaintext attacks [59] against encryption used in IPsec in ESP tunnel mode. These attacks show how traffic could be used in cryptanalysis, providing a fair amount of probable plaintext to the attacker.

An attacker starts with prediction about certain properties of the encrypted packets. Bellovin states that measurements have shown that 30-40% of network packet traffic [60] [61] is 40-byte TCP ACK-packets. If random padding is not used, analysis of packet length distribution would yield sufficient information to probably suggest the type of an encrypted packet. The attacker starts analyzing the positions of bits in the encrypted packet.

3.2.1 Probable Plaintext in the IP Header

In tunnel mode, the first part of the encrypted ESP payload is the IP header of the original datagram (IPv4). Since the position of inner IP header fields are known [28], only the guessed values of the fields will be given (see Figure 11).

The *Version* number is always 4 for IPv4 and the *Header Length* of the packet is mostly 5, followed by *ToS/DiffServ* field which generally is 10. From the length of the packet, the *Total Length* field could be determined and in case of TCP ACK packet, the value should be 28. If random padding is not used, traffic analysis could be used to determine the *Total Length* field value. Prediction of these fields results in 32 bits of probable plaintext.

We proceed by looking at values which could be guessed with some certainty. The *Flags* field and the *Fragment offset* field are generally set to zero. *Protocol* field is 6 if TCP is used as transport protocol or 11 for UDP. In case the attacker has some knowledge about the distance between the packet sender and recipient, the high order bits of the *TTL* field could also be guessed.

The rest of the packet header is the source and destination address fields. Assuming that the *Options* field is not used, then, through traffic analysis, a TCP ACK packet could be chosen for analysis. There are several scenarios for determining the address fields. First, if an ESP tunnel mode connection is established between a host and a gateway, either the *Source Address* or *Destination Address* field could be known. Since the IP address could easily be known for a host, this knowledge gives an additional 32 bit of plaintext. Secondly, if

an ESP tunnel mode connection is used between two hosts, then both the *Source Address* and *Destination Address* are known, giving 64 bits of plaintext. Alternatively, when we have an ESP tunnel mode connection between two gateways, 16-24 of most significant bits of the *Source* and *Destination Addresses* could be guessed.

3.2.2 Probable Plaintext in the TCP header and UDP header

Bellovin continues analyzing IPsec ESP in details, and follows the same methodology as used in the previous section. Traffic analysis helps us to identify encrypted TCP packets among the rest of packet traffic. Bellovin starts by identifying and guessing values of TCP header fields, the *Source Port* and *Destination Port* field could be considered random, but Bellovin claims in Section 6 in [59] that these values could be deduced from the traffic characteristics, giving an additional 16 bits of probable plaintext. *Sequence Number* should also be considered as random, but in certain cases the value could be predicted [62]. The *Acknowledgment Number* is the value of *Sequence Number* in case an acknowledgment packet is sent back. However, if the initial SYN packet sent by the client to open the connection could be identified, the *Acknowledgment Number* field is zeroed in this specific case, giving 32 bit of plaintext. Bellovin claims in Section 4.2 of [59], that some values of the *Flags*, *Window*, and *Urgent Pointer* field could be predicted. These predictions would give a total of 88 bits with some uncertainty of probable plaintext. If a UDP header is analyzed, Bellovin claims in Section 4.3 of [59] that 28 bits of probable plaintext can be guessed.

IPsec uses padding in different cases: to pad encryption data to cipher block size (ESP) and align the packet to a 4-byte word boundary. Padding could be used to provide partial traffic flow confidentiality. Since IPsec ESP padding (0-255 bytes) is added at the end of the data payload, the positions of the different fields mentioned are known and the padding does not obscure the identification.

4 ATTACKS AGAINST IPSEC

In this section, attacks against IPsec protocol in general will be reviewed. The attacks are not theoretical but proof of concepts has been implemented in most cases. At the end of Section 4 a conclusion of the effects and risks of these attacks against the IPsec configuration used in this thesis work is included.

4.1 Replay attack

IPsec's Encapsulating Security Payload [7] and IPsec's Authentication Header protocol [3] require the anti-replay service to be disabled, if manual keying is used. The explanation in Section 5 in both [7] and [3] states the following:

“If the key used to compute an ICV is manually distributed, correct provision of the anti-replay service would require correct maintenance of the counter state at the sender, until the key is replaced, and there likely would be no automated recovery provision if counter overflow were imminent.”

Ferguson and Schneier [57] gives a scenario of a possible attack if the anti-replay service is turned off. Suppose that both an AH transport mode SA (denoted SA_{AH}) and an ESP transport mode SA (denoted SA_{ESP1}) have been negotiated. Now a tunnel is established providing both confidentiality through SA_{ESP1} and authentication through SA_{AH} , but no anti-replay protection is enabled. After the data has been transmitted, SA_{ESP1} is deleted from the SAD but SA_{AH} is kept.

Later a new ESP transport mode SA is negotiated, denoted SA_{ESP2} , and the user *chooses* the same SPI value for SA_{ESP2} as used for SA_{ESP1} . The attacker now replays packets used from the first exchange. The receiver authenticates the packets through SA_{AH} , and finds the packets valid. The receiver proceeds to decrypt the packets, but the data has been encrypted by SA_{ESP1} which probably garbles the original data. The receiver believes that the traffic is authentic and presents the decrypted data, garbled data, to the application with unknown consequences.

This situation could have been avoided by authenticating the data and then encrypting it. In case several SAs are used for IPsec processing, all the related SAs should be deleted when not needed anymore. The SPI number should also be different each time a SA is created, a counter would be suitable for this purpose. Also, manual keying has its limitations in busy and larger networks, although it is easier to administer in a small scale network - but where the threshold of shifting to automatic keying is - lies outside the scope of this thesis.

4.2 CPU overload DoS attack

One of the steps in the IPsec inbound processing, is packet authentication verification, a moderately computational costly process. Section 5.2, in RFC 2401 [2] explains the steps taken when an incoming IPsec packet is processed. An attacker could misuse the MAC verification process to launch a Denial of Service attack against an IPsec endpoint by sending a large number of bogus IPsec packets to the IPsec endpoint.

Depending on the processing capacity of the system, the vulnerability for this kind of Denial of Service attack is obvious. Both ESP and AH protocols are vulnerable to such an attack, and an IPsec endpoint cannot distinguish between legal and illegal packets until it performs computation. Even if anti-replay service is

applied, packets could be sent with correct sequence numbers. Recall that the sequence number field could be known to an attacker, because the sequence number, both in AH or ESP processed packet is sent in clear text and the value is increased monotonically. Therefore the receiving endpoint, will accept the sequence number value, rendering the anti-replay service useless in this case, and forwarding the packet for authentication processing.

Touch and Yang published a paper regarding this problem [63]. The authors published their experiment measurements about throughput reduction when an IPsec endpoint is attacked. The paper handles three types of attacks. *Alg* stands for Algorithm:

- *Alg-SPI*, an attacker sending spoofed IPsec packets with incorrect IPsec SPI value
- *Alg-key*, an attacker sending spoofed IPsec packets with correct IPsec SPI value but encrypted with incorrect key
- *Alg-OK*, an attacker sending spoofed IPsec packets with correct IPsec SPI value and processed with the correct key

The attacks in the experiment were conducted against four different IPsec configurations, IPsec *HMAC MD5* authenticated *transport mode*, IPsec *HMAC SHA1* authenticated *transport mode*, *DES* encrypted IPsec *transport mode*, and *TDES* encrypted IPsec *transport mode*.

The paper shows that an attacker is able to reduce IPsec authenticated traffic throughput by 20% with *Alg-SPI* attack, 40% with *Alg-key* attack, and with 50% with the *Alg-OK* attack (see Figure 4 in [63]). In case of attacks against IPsec encrypted traffic, the throughput reduction was similar to the previous results, but the *Alg-key* attack caused a reduction of about 50% (see Figure 5 in [63]).

4.3 Sliding Window Attack

The IPsec anti-replay service is implemented through a sliding window. The window size must be a minimum of 32 packets long, but can also be larger, e.g. 64 or 128 packets long or even longer. All packets sent with sequence number falling to the left of the sliding window (low values) are automatically dropped. The same applies to packets with sequence number within in the window, but marked as already received. ESP RFC 2406, Section 3.4.3, states the following:

“All ESP implementations MUST support the anti-replay service, though its use may be enabled or disabled by the receiver on a per-SA basis. This service MUST NOT be enabled unless the authentication service also is enabled for the SA, since otherwise the Sequence Number field has not been integrity protected.”

Since the mandatory simultaneous use of the anti-replay service and authentication service is a configuration matter, IPsec is vulnerable for misconfiguration errors by system administrators. The vast number of different settings of modes, protocols, and other security settings might lead to weak security configurations. This issue was pointed out by Ferguson and Schneier in Section 3.1 in [57]. The IPsec documentation states that the implementation of the anti-replay sliding window is a local matter, but requires packets to be authenticated before updating the window. Paterson and Yau [64] have previously explained weaknesses of unauthenticated ESP tunnel mode packets encrypted in CBC mode. The same technique and configuration used by Paterson and Yau, would help an attacker to change the *Sequence Number* to the desired value.

Liang and Muradyan explain in Appendix A of [65] that the anti-replay sliding window is moved to the right once a packet with a *Sequence Number* higher than the sliding window's right edge arrives. If the update is simply made by shifting one step to the right, an attacker could send several packets with a high *Sequence Number* which will update the sliding window far to the right. This will cause incoming legitimate packets to be labeled as old or replayed, and therefore to be dropped. This would cause a total IPsec traffic stop and a Denial of Service situation. If the sliding window gets updated to the position of the incoming packet's *Sequence Number*, a single packet with a *very high Sequence Number* is sufficient for a Denial of Service attack against the IPsec endpoint. Since the packet would be updated to the far right, no legitimate packet would be accepted. Liang and Muradyan warn about this situation and point out that the sliding window should not be updated until the packet passes an *acceptance test*. There is no explanation about what a suitable *acceptance test* is, but probably authentication verification is meant. Since the 3GPP specification requires that ESP mode with authentication must be used, this class of attack will **not** succeed in this configuration.

4.4 Attacks against unauthenticated ESP traffic in CBC mode

4.4.1 Bellovin's attack

Bellovin published an article [59] about an attack which defeats a specific configuration of IPsec encryption. To succeed with the attack there are some prerequisites: IPsec ESP **without** authentication is used and for encryption CBC mode [40] [6] [46] is applied. CBC encryption mode is vulnerable to a *bit-flipping attack*, see Section 2.3 in [64], which enables an attacker to introduce controlled changes to the decrypted cipher text. In this attack, the attacker needs to be able to send packets through the same IPsec gateway and should be able to capture, delete, and modify packets.

The attack works as following: Eve, denoted E , wants to read the secret sent by Alice, denoted A , to Bob, denoted B (see Figure 19). Therefore E captures the packet sent from A to B , denoted P_A . E also sends an UDP packet through the same IPsec connection and captures the packet after it has been encrypted, denoted P_{E1} . Now E cuts out the encrypted TCP header and the encrypted secret from P_A . E also cuts out the IP header, ESP header and encrypted UDP header from his first packet P_{E1} and constructs a new IPsec packet.

E sends P_{E2} through the IPsec tunnel and the receiving gateway will start decrypting the data. This will probably cause a decryption error in the first cipher block, the TCP header of A 's original packet, P_A , but because of the self healing property of CBC mode³, the receiving gateway will proceed and correctly decrypt the rest of packet. Since the UDP header of P_{E2} is constructed by E , the decrypted packet could be sent to an IP address controlled by E . Authentication of IPsec packets stops this attack.

³ If one block of ciphertext is altered, the error propagates for at most two blocks.

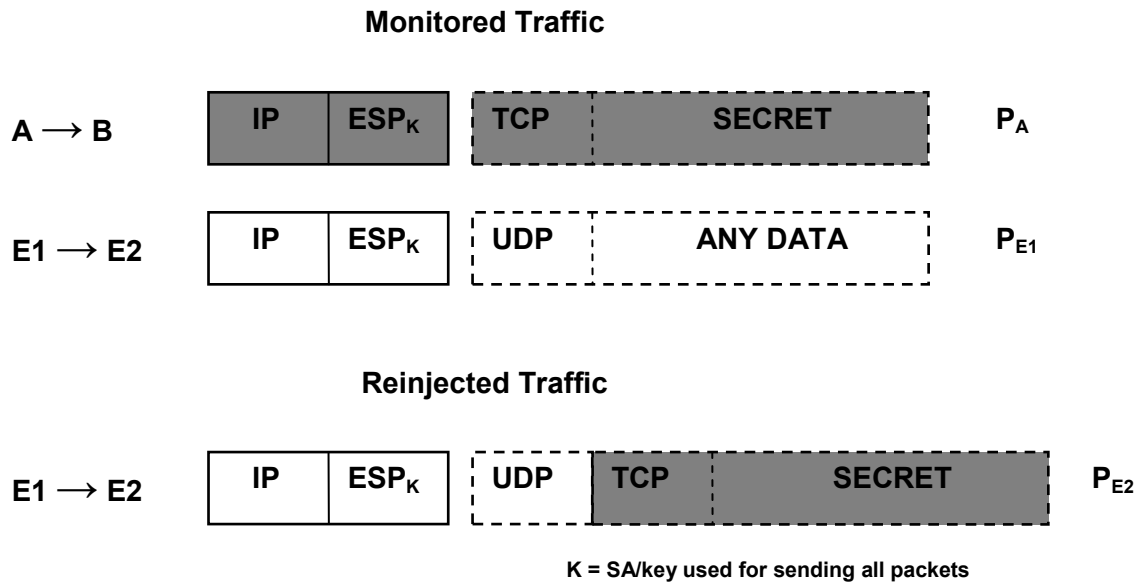


Figure 19: Modification of an IPsec packet in Bellovin's attack.

4.4.2 Paterson's and Yau's attack

Paterson and Yau published a paper [64] about serious attacks against IPsec that would enable an attacker to decrypt IPsec traffic due to certain configurations. Their research is a further development of Bellovin's attack [66]. A proof of concept was implemented and vulnerability warnings were issued worldwide afterward [67] [68]. There were several factors that made the attacks successful: the optional use of encryption without authentication, the use of CBC encryption mode, faulty IPsec inbound processing implementation, and some properties of an ICMP packet reply.

They presented three different attacks that could be launched: attacks based on *destination address rewriting*, attacks based on *IP options processing*, and attacks based on *protocol field manipulation*. The general ideas and methods of the attack will be described, for detailed information we refer to [64].

An assumption to succeed with the attack is that no security policy check is made after IPsec inbound processing for the inner IP header. However, RFC 2401 mandates that an IPsec implementation should consult the SPD after IPsec processing. This faulty IPsec implementation existed in Linux kernel release 2.6.8.1.

4.4.2.1 Attacks based on destination address rewriting

There are two versions of the attack, a 64-bit cipher block and 128-bit cipher block version. Only the 64-bit version will be described in this brief introduction. The IPsec traffic is sent between two gateways running ESP in tunnel mode with CBC encryption mode. An assumption is made that the attacker knows the destination address of the inner IP packet (see Figure 11). Recall that bit flipping in CBC for the first cipher block introduces controlled changes in the decrypted second cipher block, see Section 2.3 in [64].

Phase 1

An attacker captures a packet from the traffic he wants to decrypt (see Figure 20). The first 32 bit of second cipher block C_2 are masked with the *attacker's IP address* and the assumed known *destination address*. This would cancel out the original IP *destination address* and the decrypted cipher block would then have the *attacker's IP address* as destination address. Also the first half of cipher block C_2 (32 bit) are changed but to a random value. A problem will arise once C_2 is decrypted to plaintext, P_2 . The plaintext P_2 will contain the TTL, Protocol, IP Header Checksum and Source Destination fields. Those values must be valid to successfully reroute the encapsulated IP packet to the desired IP address. The *TTL* value should be large enough to reach the destination. The *Protocol* Field could be arbitrary because the IP destination is controlled by the attacker who handles the IP processing. The *IP Header Checksum* must be valid to not be dropped, and *Source Destination* should be routable.

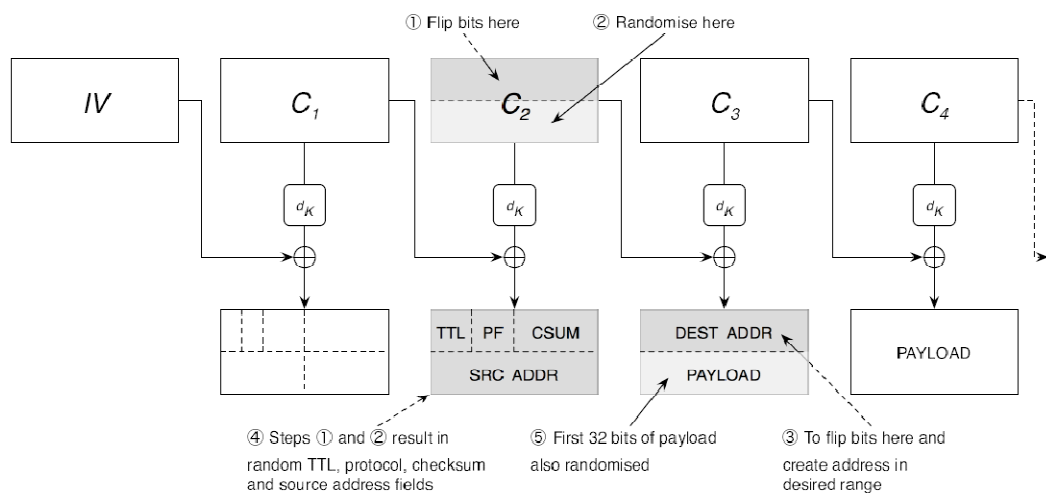


Figure 20: Phase 1 modification of IPsec packet in Paterson's and Yau's attack [64]⁴

The attack succeeds once a decrypted IP packet is routed to the *attacker's IP address*. The attacker repeats the alteration of the IPsec packet with different random value in C_2 and injects it within the IPsec traffic until he succeeds. The paper shows that success probability for the attack for each new constructed IP packet with random value in the last 32 bit in C_2 is roughly 2^{-17} . The major obstacle in the attack is succeeding to decrypt a valid value for the IP Header Checksum field. The paper estimates that 2^{-16} is the success probability to get a valid Header Checksum field. It is calculated that 2^{17} iterations of the attack would give a 60% success probability.

⁴ This figure appears here with the permission of Professor Kenneth G. Paterson

Phase 2

Once we succeed in phase 1, we reuse the information to be able to decrypt any data encrypted by the same SA. We chop the encrypted version (see Figure 21) of IPsec packet arriving to our chosen destination address into several blocks C_1 , C_2 , C_3 , ..., C_{q-2} , C_{q-1} , C_q .

The blocks C_1 , C_2 , C_3 consists of the encrypted IP and ESP headers and the blocks C_{q-2} , C_{q-1} , C_q constitutes of the ESP tail. We insert $(q-6)$ blocks of any cipher blocks encrypted by the same SA in-between. Once our modified packet is reinjected in the IPsec traffic, it will get decrypted and sent to our chosen IP address. A lot of details have been omitted and the paper should be consulted for more detailed information.

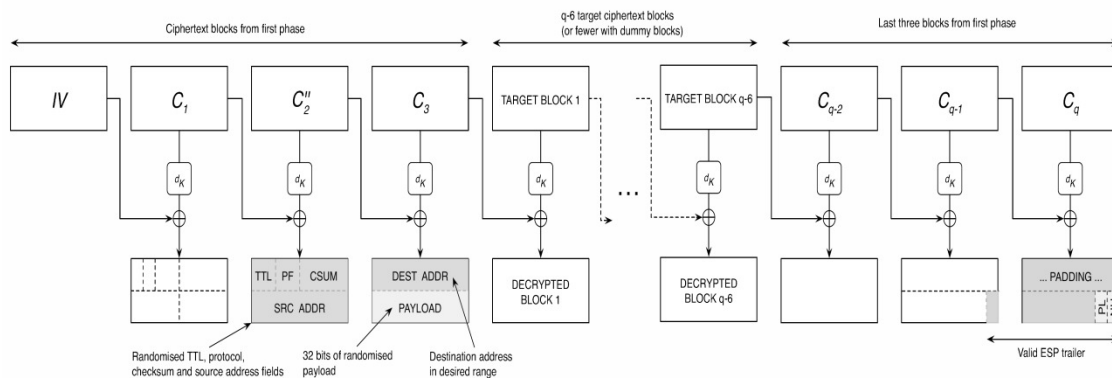


Figure 21: Phase 2 modification of IPsec packet in Paterson's and Yau's attack [64]⁵

4.4.2.2 Attacks based on IP options processing

In the second attack outlined by Paterson and Yau, the Internet Control Message Protocol (ICMP) is used to defeat the IPsec encryption.

Phase 1

The attacker starts with modifying the `Header Length` field of the inner datagram of a captured IPsec packet (Figure 20) to a value higher than 5 by flipping a few bits, and then randomizing the last 32 bytes of C_2 . The modified packet is resent to the IPsec gateway.

Once the inner datagram is processed by the IPsec gateway, the first bits of the IP payload will be interpreted as `IP options bytes`. Then with a high probability, an ICMP "parameter problem" error message would be generated. The ICMP message will contain the IP header of the inner datagram and segments of the payload. If an attacker is able to capture the ICMP message, he will be able to obtain plaintext information from the IPsec.

Randomizing the last 32 bytes of C_2 , will probably render in incorrect decrypted IP header values for, `TTL`, `protocol`, `header checksum` and `source address`, and the packet would probably be dropped

⁵ This figure appears here with the permission of Professor Kenneth G. Paterson

before any `IP Options` processing takes place. This means that ICMP packets will rarely be generated. In case an ICMP message is generated, it will probably be routed to an unknown source address if ICMP traffic is allowed through IPsec gateway. The attacker is thought to be able to capture the traversing ICMP packet. Even though the mentioned difficulty with correct header checksum, iterating the attack with new random value in the C_2 sufficiently often will achieve a sufficient success rate, as proved in the paper.

Phase 2

Once an ICMP has been captured, the attacker would be able to capture about 64 bits of IPsec payload plaintext. The attacker uses the same header causing the ICMP packet generation but inserting the chosen cipher blocks after the ESP payload header. The attacker will in this way receive decrypted cipher blocks through ICMP packet.

4.4.2.3 Attacks based on protocol field manipulation

This attack is carried out by changing the value of the `Protocol` field in the inner IP header, through bit-flipping, to one that is not supported by the end host receiving the inner datagram. This action will generate an ICMP “protocol unreachable” error message with the same properties as in the previous attack, Section 4.4.2.2, containing parts of the decrypted payload. The ICMP packet will in this case be generated by the end host, not by the security gateway. To succeed with the attack, the source address must be changed to be routed back to the attacker. The header checksum must also be correct in order to not be dropped. The same iteration and block cipher copy-and-paste techniques as used in the previous attack, applies also here. For more detailed information, see Section 5 in [64].

4.5 IPsec attacks summary

The attacks mentioned earlier pose a serious threat against some IPsec configurations. In this thesis work, ESP tunnel mode with authentication is used as specified in the 3GPP specification TS 33.210 [1].

The replay attack described in Section 4.1 will not work because AH is not used to provide authentication, but rather ESP is. The same ESP SA will provide both authentication and encryption. Also, once a new SA is created, a new SPI number must be given which preferably has not been used recently. Remember, the SPI field is 32 bit long. According to the RFC as explained in the section, anti replay should be turned off if manual keying is used. The solution with manual keying is temporary and automatic key management will be used in the future. The attack is **not** a threat. However, if **anti replay is turned off**, the system is vulnerable to “regular” replay attacks, i.e., resending captured packets at later stage.

The CPU overload DoS attack described in the Section 4.2 is a real threat to some IPsec architectures. In this work, BITW IPsec architecture is used, i.e., IPsec processing is done by a dedicated hardware which works on wire speed and the attack will **not** work. This attack poses a threat in case the IPsec processing is done by software on a multipurpose CPU.

The Sliding Window Attack described in Section 4.3 works only if anti-replay is turned off. The temporary solution with manual keying should avoid anti-replay protection. If anti-replay is not used, this attack is a threat. However, since automatic key management and anti-replay protection will be used this attack is **not** a threat.

The attacks against unauthenticated ESP traffic in CBC mode described in Section 4.4 do **not** pose a threat against the IPsec configuration used in this thesis. The 3GPP specification [1] requires authentication to be used and the attacks would fail in that case. It is crucial to have authentication protection, since such attacks are among the most serious attacks against IPsec. A proof of concept has been implemented, which led to issuance of worldwide warnings about the risks of such attacks.

In addition to those attacks mentioned above, an attacker might make use of program specific implementation vulnerabilities to attack a system. If an IPsec gateway generates log files and no restriction is placed on the volume of these log files, an attacker might try to generate large amounts of log data to use all available storage space or to exhaust the system resources in some way. An attacker might also use the fragmentation properties of IP to force the discard of legitimate IPsec packets. By sending fake IPsec packet fragments, the fragments will be saved in the fragmentation reassembly memory area, which at some point will overflow which will might lead to legitimate IPsec packets being discarded.

5 METHOD

IPsec packets might be modified during transport because of natural occurring transmission errors. On an 1000BASE-T/1000BASE-X Ethernet link as our incoming data will be received on, a 32 bit CRC (Cyclic Redundancy Check) will be able to detect $1 - 2^{-32} \approx 99.99999998$ % of all errors [69]. If we have 300 Mb/s ESP traffic with SHA-1/AES-CBC in tunnel mode, assuming we sent the maximum amount of data possible in each Ethernet frame, i.e., 1500 bytes, we will receive around 25000 Ethernet packets/s. The probability of getting a bit error in a coaxial cable is 10^{-9} and fiber optics is 10^{-12} [70]. If we make the unrealistic assumption that every Ethernet frame we receive contains a random error, the time until we receive an incorrect Ethernet frame with a correct CRC is:

$$\frac{1}{25000 \times 2^{-32}} \approx 171799 \text{ seconds} \approx 48 \text{ hours}$$

This shows that it is very unlikely that a transmission error will corrupt an IPsec packet, but a hardware failure could generate incorrect packets with correct CRC values. A malfunctioning switch might corrupt a packet before adding the CRC. The receiver will not detect such errors by verifying the CRC, but hopefully these errors will be detected in later stages of the packet processing. A third party might also intentionally modify IPsec packets in an effort to attack a system or to gain information. These modifications will cause the IPsec packet to fail probably authentication (or at later checks).

Therefore it is important to notice that **some failures will occur for reasons other than an attack**. Hence it is important to look in detail at both the rate and number of such failures and to exploit other information to determine if there really is an attack in progress.

The IPsec configuration used in this project uses **ESP Tunnel Mode with authentication protection** as specified in Sections 5.3.1, 5.3.2, 5.3.3, and 5.3.4 in the 3GPP specification TS 33.210 [1]. Figure 22 shows an overview of the logical setup used in this thesis for detecting intrusions against IPsec. The source labeled `Traffic generator` sends IPsec packets to the `Traffic destination`. The network processor on the Ericsson Ethernet Interface board processes the received packets through the IPsec block before forwarding them to the `Traffic destination`. If a modified packet is processed, the IPsec processing will fail and the network processor will forward the modified IPsec packet together with the error codes to the `Analyzer`.

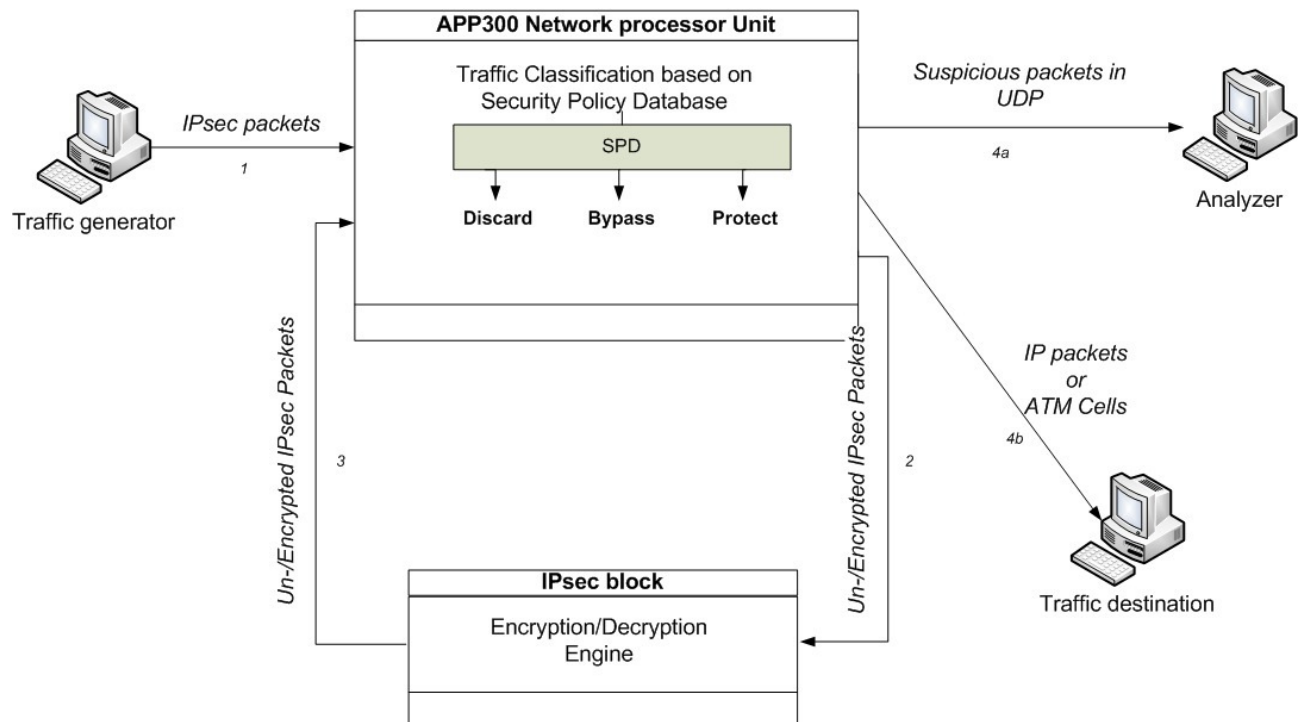


Figure 22: A logical overview of the Intrusion Detection Setup for the Ericsson Ethernet Interface Board

In this project, the classification engine in the network processor was modified to send the interesting parts of the modified IPsec packets (see Table 8) to the *Analyzer* for further analysis, instead of simply discarding the packets. The encrypted payload of a modified IPsec packet is not interesting for our implementation of the *Analyzer*, since it will consume bandwidth in the network processor and it will not yield any further information for analysis. This project is divided in two main parts. Since the IPsec hardware and the software implementation are not ready yet at this date, a simulation of the NP in the Ericsson Ethernet Interface board will be used. In the first part of the project, steps 1, 2, and 3 (see Figure 22) of the data flow diagram are simulated using the LSI System Performance Analyzer (SPA) software (see Section 5.2).

The second part of the project will simulate the entire process, using an IPsec software implementation instead of hardware implementation. A **second** traffic generator (see Section 5.3.1) developed based upon the first phase, will be used to specify and simulate the data sent from the board to the *Analyzer*, and an IPsec stack implementation will be modified to simulate the behavior of the NP in the board.

5.1 Different scenarios of IPsec processing

In each case, the error codes generated by the NP could either be due to hardware failure/bad IPsec processing before sending to the Ericsson Ethernet Interface board, or an attacker. However, the knowledge about what type of failure occurs can be useful to investigate the possible reasons. In this section we explain the possible classes of failure and what resources the attacker needs to access to avoid detection.

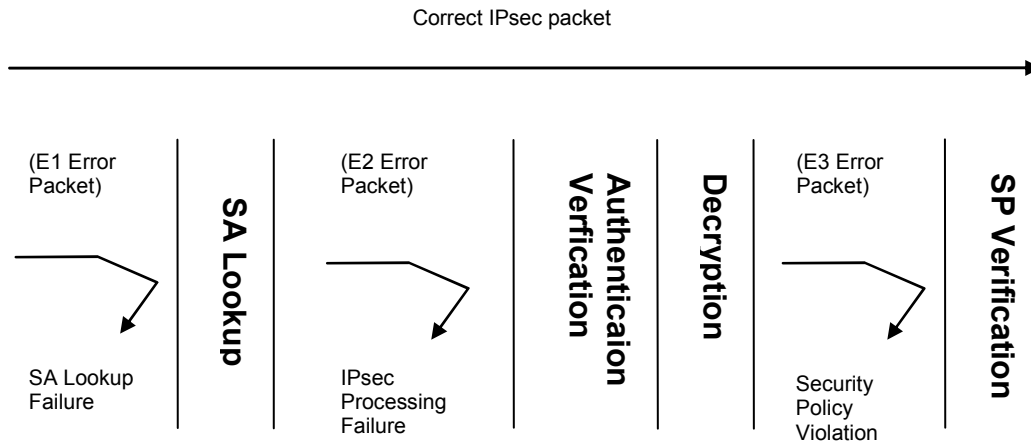


Figure 23: The different steps for an incoming IPsec packet to traverse through an IPsec gateway

Referring to the IPsec processing diagram in Figure 23, there can be the following types of failure:

- SA lookup failure
- IPsec processing failure
- Security Policy violation

The attacker is able to send a “correct” IPsec packet only when the packet is able to pass all checks. In the following discussion, we will refer to the letters in Figure 24 to identify the parts of the network the attacker can access.

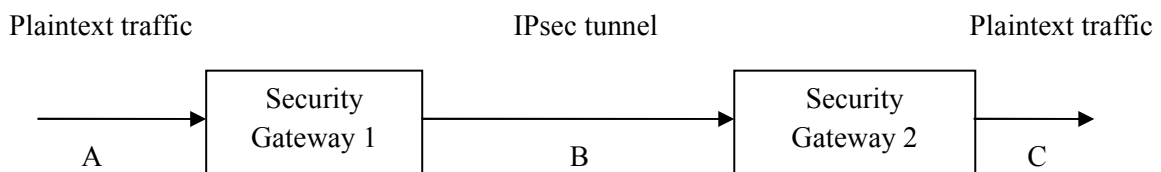


Figure 24: View of a network protected by two IPsec gateways

5.1.1 SA lookup failure

Detection of an SA lookup failure reveals that the attacker can inject packets at B (see Table 5). However, in the first case, the attacker has no knowledge of valid SAs, but can only guess. Unless the attacker has guessed valid SA, a SA lookup failure will occur.

In the second case the attacker is able also to sniff the packets at B, then the attacker could create a packet with a valid SA and avoid detection at this stage, but such a packet will most probability be detected at next stage, “IPsec processing failure”.

Table 5: Attack scenarios during SA lookup failure

Case	Attacker's capability	Attack scenario	Detection by Analyzer at this stage
1	Inject packets at B	Attacker injects a packet at B, trying to guess a valid SA.	Yes
2	Inject packets at B Sniff packets at B	Attacker sniffs the traffic at B and injects a packet with valid SA selectors value at B.	No

5.1.2 IPsec processing failure

Detection of an IPsec processing failure reveals that the attacker can inject and sniff packets at B (see Table 6). The attacker is able to pass the SA lookup step simply by sniffing the packets passing through B and looking at the SA selectors: Destination Address and IPsec Protocol (Protocol field) in the IP header, and SPI in the ESP header. The attacker needs only to read these fields, transmitted in plaintext, create a new packet with the correct selectors for a valid SA, and inject the packet. However, the attacker has no knowledge of the authentication key for the SA, therefore the packet will be discarded at the authentication verification step, causing an IPsec processing failure.

Table 6: Attack scenarios during IPsec processing failure

Case	Attacker's capability	Attack scenario	Detection by Analyzer at this stage
1	Inject packets at B Sniff packets at B	The attacker sniffs the traffic at B and injects a packet with valid SA selectors at B but incorrect authentication data.	Yes
2	Inject packets at A (Sniff packets at B)	The attacker injects a plaintext packet at A, which will be encapsulated in an IPsec packet with correct authentication data. The attacker could also sniff the authenticated packet at B to gain information about the authentication key.	No

5.1.3 Security policy violation

The detection of a security policy violation reveals that the attacker can inject and sniff packets in A (see Table 7) and sniff packets in B (not necessary), but is unaware of the security policy set at the Security Gateway 2. By injecting packets at A, the attacker will have the packets IPsec processed by Security Gateway 1, since it shares the keys with Security Gateway 2. Therefore the packets will pass the authentication and decryption step. The attacker could use this attack to gain information on the encryption key used for the SA, comparing the plaintext packets injected at A with the (same) encrypted packets, sniffed at B (if possible). If the attacker is able to find the encryption key before detection and rekeying, he could decrypt protected traffic. Therefore, detection time and the time before rekeying must be configured properly in order to avoid this situation.

Table 7: Attack scenarios during security policy violation

Case	Attacker's capability	Attack scenario	Detection by Analyzer at this stage
1	Inject packets at A (Sniff packets at B)	The attacker injects a plaintext packet in A, which will be encapsulated in an IPsec packet with correct authentication data. The attacker could also sniff the authenticated packet at B to compare cipher and plain text pairs for cryptanalysis.	Yes
2	Sniff packets at A Inject packets at A (Sniff packets at B)	Attacker sniffs the traffic at A and injects a plaintext packet at A, which will be encapsulated in an IPsec packet with right authentication and will have the same selectors as the "normal" packets. The attacker could also sniff the authenticated packet at B to compare cipher and plain text pairs for cryptanalysis.	No

5.1.4 Correct IPsec packet

To send a correct IPsec packet, the attacker needs to inject packets in A, and sniff packets in A. Assuming he is able to pass all the previous steps, in order to pass the security policy verification step, the attacker needs to eavesdrop the security policy selectors of packets to be IPsec processed. Some of these selectors are the original source and destination addresses, and transport layer protocol, which, in case of tunnel mode, will be encrypted when the packet is processed by Security Gateway 1. Therefore, the attacker also needs to sniff the packets at A, intercept the security policy selectors, create a new packet with the correct selectors for a valid security policy, and inject the packet at A. Note that being able to sniff at C would also reveal which packets make it through all the way, hence it is possible to learn the correct security policy selector values.

5.2 LSI System Performance Analyzer

The first part of the project is performed in a simulated environment. The LSI System Performance Analyzer (SPA) [71] version AG_NP-3.9.0.56, sample configuration file `app3_ipsec_ipv4_lb.xml` is used to simulate two IPsec gateways communicating, by looping back the data in the same IPsec gateway, loopback mode, see Figure 25.

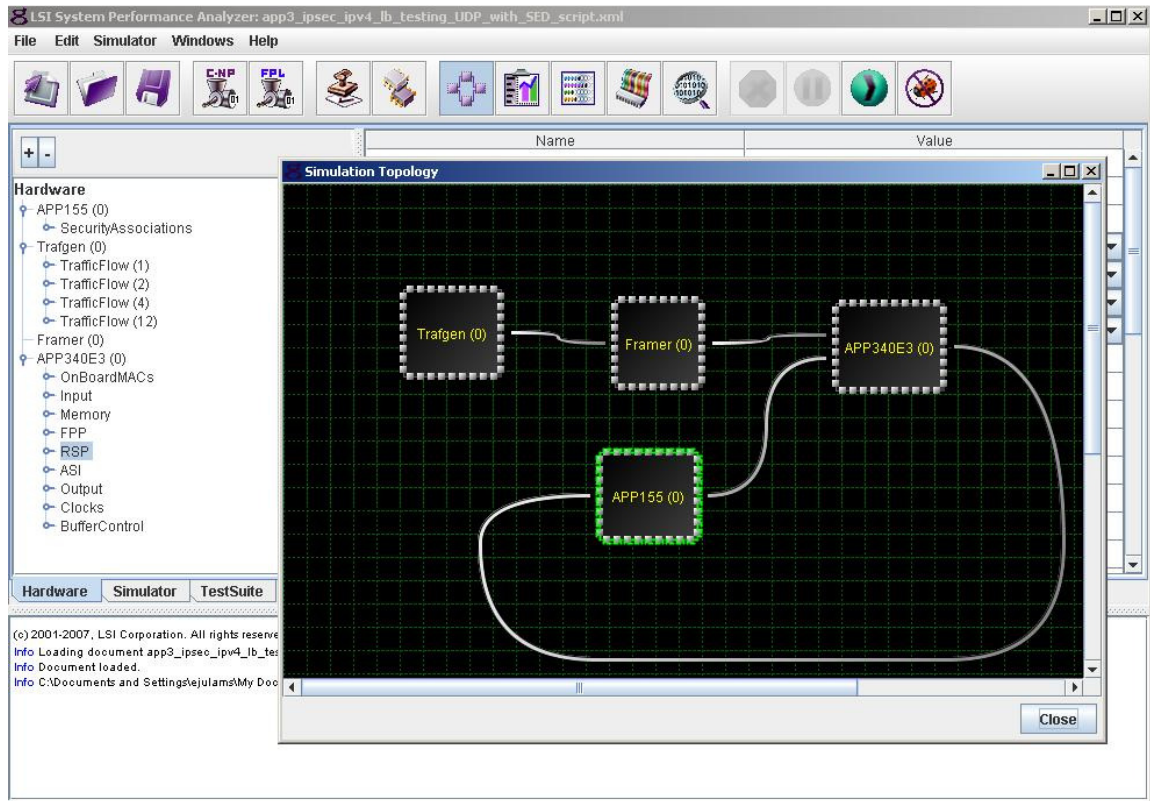


Figure 25: Screenshot of the LSI System Performance Analyzer

The following steps are executed to simulate two IPsec gateways (SPA and sample configuration file specific):

1. A traffic generator sends an **IP packet** to a destination through the IPsec gateway (the APP340 network processor (NP) and APP155 IPsec coprocessor)
2. The packets are split into protocol data units (PDUs) of 64 bytes for internal routing within the network processor.
3. The blocks of the **IP packet** are received at input port 2 in NP
4. The blocks are passed to the classifier which executes FPL code to process the data in two passes:
 - In Pass 1, the blocks are identified and reassembled
 - In Pass 2, the **IP packet** is classified and a Destination ID (DID) is derived based on the Destination Address field in the IP header, to identify IPsec mode and protocol, and

on the `Next Header` field, to identify if inbound or outbound IPsec processing is necessary. Details of this DID are explained in Section 5.2.4

5. The IP packet is passed to the NP's Traffic Manager, which executes the Stream Editor (SED) script (written in C-NP) linked with the chosen DID. The script prepends the SPP Transform Internal Header to the packet for IPsec processing
6. The **IP packet** is sent to the IPsec block through output port 4
7. The IP packet is encrypted and authentication data is added (IP → IPsec) and the **IPsec packet** is sent back to NP input port 4
8. The **IPsec packet** is classified in the NP and sent to loopback port 3 to simulate an incoming IPsec packet
9. The incoming **IPsec packet** is received from input port 3 and classified as in steps 4 and 5
10. The **IPsec packet** is sent to the IPsec block through output port 4
11. The IPsec packet is decrypted and authentication is verified (IPsec → IP) and the **IP packet** is sent back to NP input port 4
12. The **IP packet** is classified in the NP and sent to port 2 to its final destination

5.2.1 Classification of modified IPsec packets

As mentioned earlier, if a modified packet is processed by the IPsec block, only a non-fatal error will occur, since fatal errors only occur because of an incorrect SED script executed to add SPP processing header (see Section 7.3.1 in [72]). The IPsec block will prepend a result header to the IPsec packet (Figure 4) setting the relevant error bits (Figure 6). The Classification Engine (CE) is programmed in a language called Functional Programming Language (FPL). The CE FPL code checks the two most significant bits in the SPP result header that was appended to the IPsec packet. In our case, the CE FPL code has been modified such that if a non-fatal error occurs (i.e., the error bits are 01), see Table 1, then instead of discarding the packet, it should be sent to the Traffic Manager (TM). The traffic manager will in turn forward the packet to the output port, which will result in its being sent to the Analyzer. The CE sends the packet to the TM by executing the `fTransmit()` function [22], which is used to specify:

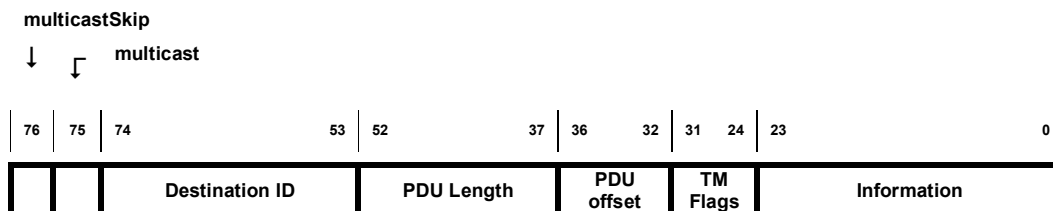


Figure 26: Chart of fields in the `fTransmit()` function

Destination ID	the DID assigned to the IPsec packet points to the SED script to encapsulate this packet in an UDP packet destined to the Analyzer. See details in Section 5.2.4.
PDU length	specifies the length of the packet sent from the CE to the TM. The PDU length is different depending on the error type generated.
PDU offset	this field specifies the starting byte to read the PDU
Information	this field can be used to pass information to the TM (see Figure 26). We have used this field to pass a counter value, which is used and updated in the FPL code of the CE. This value is copied into the Identification field of the IP header through the SED script executed in the TM

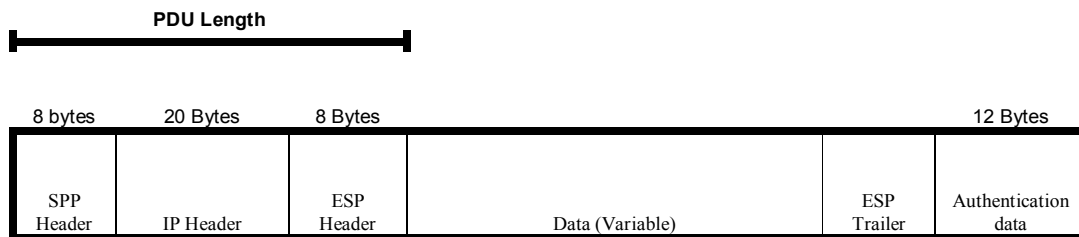


Figure 27: Only the indicated data of the IPsec packet is sent from the Classification Engine to the Traffic Manager in case of an IPsec processing failure (E2 error packet)

5.2.2 Traffic Management of modified IPsec packets

Once classification is done, the TM receives the packet and encapsulates it in a UDP packet destined to the Analyzer. To do this a SED script is executed which prepends an Ethernet header, IP header, UDP header, and Analyzer packet ID header to the portion of the packet sent from the CE to the TM, see Figure 28. The Analyzer packet ID header is used by the Analyzer to identify the type of packet received, see Table 8.

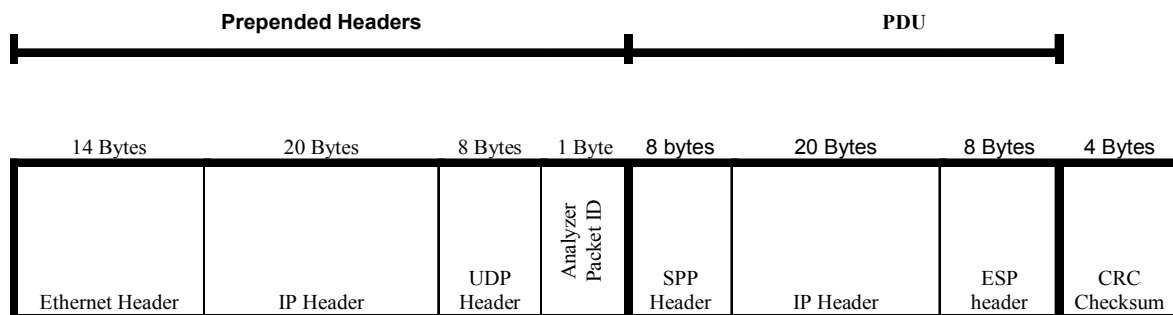


Figure 28: The prepended headers of the PDU sent to the Analyzer in case of an IPsec processing failure (E2 error packet)

Table 8: The different packet types sent to the Analyzer

Analyzer ID	Packet type	Description	UDP packet payload	Payload size
0x01	E1 Error Packet	Packet with SA lookup failure data	Analyzer ID + IP header + ESP header	$1 + 20 + 8 = 29$ byte
0x02	E2 Error Packet	Packet with IPsec processing failure data	Analyzer ID + SPP Header + IP header + ESP header	$1 + 8 + 20 + 8 = 37$ byte
0x03	E3 Error Packet	Packet with IPsec violation data	Analyzer ID + IP header	$1 + 20 = 21$ byte
0x04	IPsec packet	Packet with SA selectors of sniffed IPsec packet	Analyzer ID + IP destination address + ESP SPI number	$1 + 4 + 4 = 9$ byte

The SED script must execute quickly and complete processing during two “global pulses”. “When writing a C-NP script, you must know how much time it takes to process a block and maintain line rate. In the Traffic Manager, this duration is called the global pulse and refers to the unit of processing time available for each of the processing units, or pipe stages. The global pulse is configurable. APP300 scripts must complete within 23 clocks (1 global pulse) for full line rate, except in the SED compute engine that can have up to 46 clocks (2 global pulses) for full line rate due to its double compute engine structure” (Section Compute Engine Overview, Compute Engine Timing in [73]). Our SED script completes all modifications in 38 clocks. The script calculates the IP header checksum, but does not calculate a UDP checksum, it is simply set to zero, due to processing time limitations mentioned earlier. This is allowed because of the optional use of the UDP checksum field, as specified in [74]. All the values for the Ethernet header, IP header, and UDP header are known in advance and can be statically specified in the SED script, e.g.:

- The Ethernet frame source MAC address (Ericsson Ethernet Interface board), and destination MAC Address (Analyzer) are known
- The IP header checksum can be calculated in advance since all IP header values are known and fixed in value, e.g. source IP address (Ericsson Ethernet Interface board), destination IP address (Analyzer), Protocol (IP), and packet length
- The UDP header values are also known, the source port (Ericsson Ethernet Interface board), destination port (Analyzer), and length are all fixed value; and the checksum is zero since calculation of the UDP checksum is optional

5.2.3 Creating a SED Script in the SPA

To be able to execute the necessary SED script, a SED script entity must be created in the SPA. Using the graphical user interface in the simulator we navigated to APP340E3 → RSP → SEDScripts and created a new SED script. This requires that we specify the `id` (SED script number) and `file` (location) values.

5.2.4 Creating a DID in the SPA

To create a DID entity in the SPA, we navigate to APP340E3 → RSP → DestinationIds and create a new DID. Figure 29 shows the different values that can be set. Here we will mention only the values of interest in this case:

<code>headerDelta</code>	specifies the number of bytes prepended to the PDU
<code>Id</code>	specifies the ID number of the DID
<code>queueId</code>	specifies the queue which will be used in the TM. A queue has a priority and port ID
<code>sedScriptId</code>	specifies the ID number of the SED script to be executed
<code>sedScriptParms</code>	sets the <code>SED_param_block</code> in the SED register file map, see Appendix D. In this field we specify the Analyzer packet ID header value. The SED script will read this value from the SED register file map and write it into the prepended Analyzer packet ID header

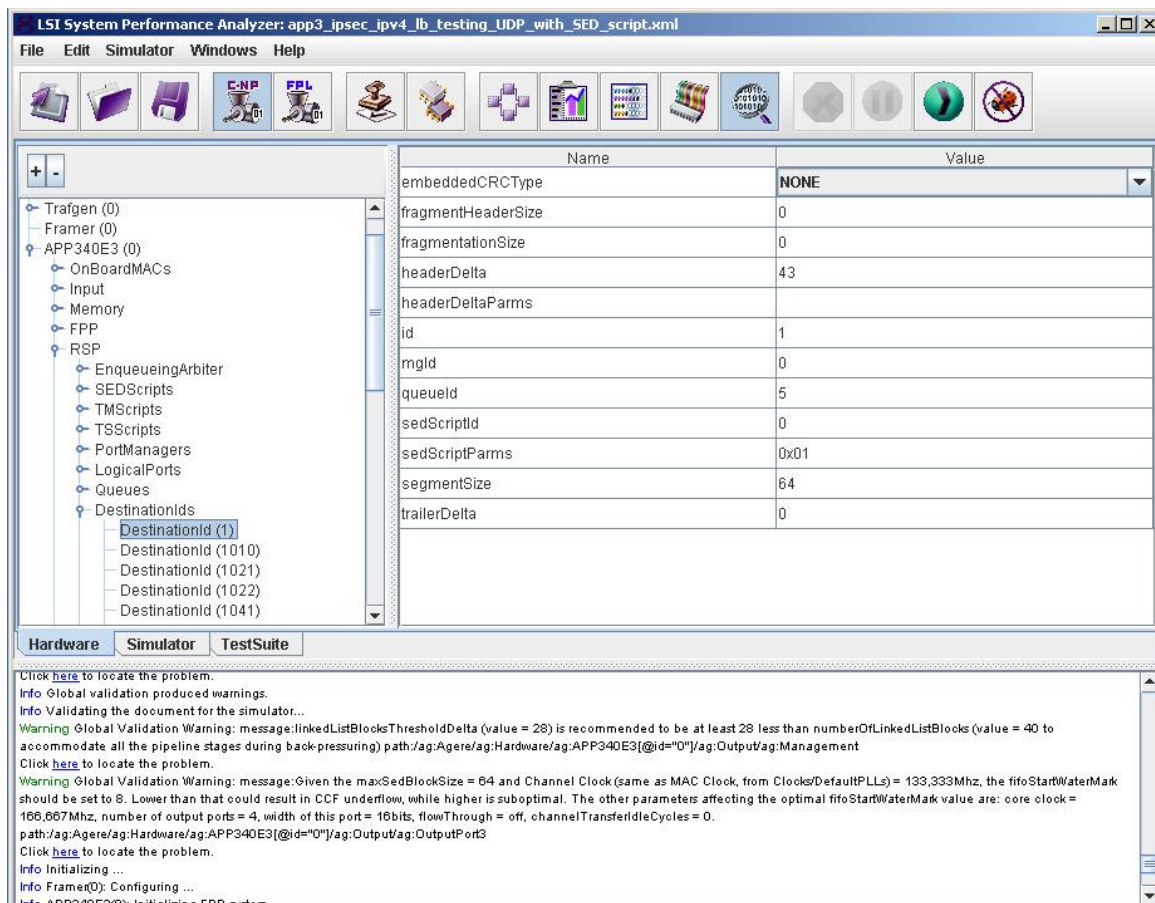


Figure 29: Creating a Destination ID (DID) in the SPA

5.3 Software IPsec stack implementation

5.3.1 Traffic Generator

To simulate different types of data used in both verification of the Analyzer’s functionality and used later for simulation, a traffic generator was developed. The traffic generator reads data in hexadecimal format from standard input or a file and sends UDP packets to the specified IP address and port number. The command usage is shown executing the traffic generator with `-h` flag. In the data file, a user can specify, e.g., the payload, the number of packets to send of the specific payload format, and the time to wait before sending the next UDP packet.

```
134.138.45.182 - PuTTY
ejulams:~/analyzer/generator> ./tg -P test_case_2-ctr_pkts.txt
Client is sending to 127.0.0.1:1700

Number of packets to send of this format 1
Payload: 0x030000000000000003E8

Sending packet 1

Program has been running in 0 seconds

Waiting for 120 seconds

120 > 119 > 118 > 117 > 116 > 115 > 114 > 113 > 112 > 111 > 110 > 109 > 108 > 107 > 106 > 105 > 10
4 > 103 > 102 > 101 > 100 > 99 > 98 > 97 > 96 > 95 > 94 > 93 > 92 > 91 > 90 > 89 > 88 > 87 > 86 >
85 > 84 > 83 > 82 > 81 > 80 > 79 > 78 > 77 > 76 > 75 > 74 > 73 > 72 > 71 > 70 > 69 > 68 > 67 > 66
> 65 > 64 > 63 > 62 > 61 > 60 > 59 > 58 > 57 > 56 > 55 > 54 > 53 > 52 > 51 > 50 > 49 > 48 > 47 > 4
6 > 45 > 44 > 43 > 42 > 41 > 40 > 39 > 38 > 37 > 36 > 35 > 34 > 33 > 32 > 31 > 30 > 29 > 28 > 27 >
26 > 25 > 24 > 23 > 22 > 21 > 20 > 19 > 18 > 17 > 16 > 15 > 14 > 13 > 12 > 11 > 10 > 9 > 8 > 7 >
6 > 5 > 4 > 3 > 2 > 1 >

Number of packets to send of this format 1
Payload: 0x030000000000000007D0

Sending packet 1

Program has been running in 122 seconds

Waiting for 120 seconds

120 > 119 > 118 > 117 > 116 > 115 > █
```

Figure 30: Screenshot of the packet traffic generator executing

5.3.2 Analyzer program

As mentioned earlier, to simulate incoming packets as the one generated during the simulation part Section 5.2, a traffic generator was developed Both the Analyzer program and traffic generator has been developed in NetBSD 3.0.0 operating system environment and compiled by gcc version 3.3.3 (NetBSD nb3 20040520).

5.3.2.1 The logic of the Analyzer

The Analyzer has been designed around the concept of Signal to Noise ratio (S/N). Discarding erroneous packets at each stage of the IPsec processing (see Figure 23) can be compared to applying a filter to the incoming signal. The signal part of the ratio consists of the correctly processed IPsec packets, whereas the noise part consists of hardware failure/attack packets which generated errors. After passing a stage in the IPsec processing, the total S/N ratio for IPsec processing (see Figure 23) increases, since the incorrect packets detected at the previous stage have been discarded.

This motivates the use of a sliding window, to monitor IPsec traffic per each stage. However, the final alert decision should be based on combination of alerts from multiple-size sliding windows per stage, because the size of the sliding window will affect accuracy and detection time:

Short sliding window

Advantage: shorter reaction time

Disadvantage: increased false positive rate

Long sliding window

Advantage: decreased false positive rate

Disadvantage: longer reaction time

In our prototype implementation of the Analyzer, two sliding windows per IPsec processing stage and per board (described in Section 5.1) will detect abnormalities by monitoring both the error and normal packets. A shorter sliding window SW1 (see Figure 31 and Figure 32) is implemented to shorten the detection time. A longer sliding window SW2 is implemented to have more data to analyze and is able to react more accurately. A scheme of alert situations is defined later on.

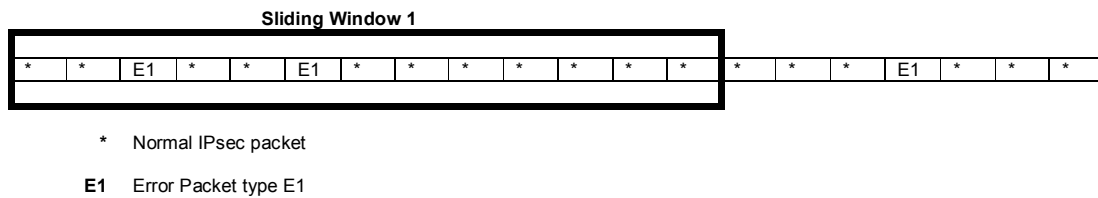


Figure 31: An illustration of sliding window 1 to monitor E1 error packets

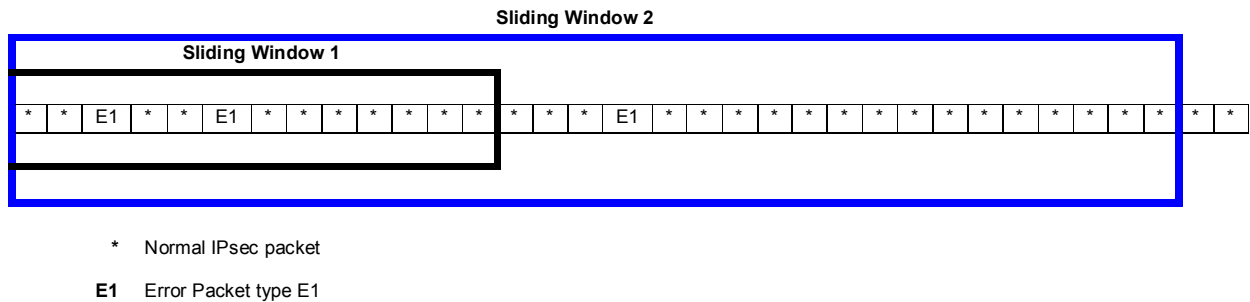


Figure 32: An illustration of sliding window 1 and 2 implementation to monitor E1 error packets

There is currently no statistics about IPsec failure rate using the actual Ericsson IPsec implementation. Therefore suitable sliding window sizes are not determined, but are easily configured through a configuration file. Board specific sliding windows will keep track of E1 and E3 errors whereas SA specific sliding windows will keep track of E2 errors. A SA is identified by the SPI, IPsec mode (ESP), and IP destination address (IPsec gateway). All these data are available per IPsec packet.

Table 9: The probabilities of different type of errors received from the board

Error Packet	Probability Value	SA specific
E1	Unknown	No
E2	Unknown	Yes
E3	Unknown	No

Every SW (sliding window) might generate two different threshold alerts. For instance *SW1* might generate *SW1 threshold 1 alert* if number of error packets in sliding window 1 is higher than the allowed threshold 1. Threshold Alert 2 is set to be higher, representing a more serious situation than Threshold Alert 1. Table 10 and Figure 33 illustrate the different cases when the Analyzer program should alert the operator.

Table 10: The different cases when the Analyzer program need to alert the operator

Sliding Window 1	Sliding Window 2	Note
Threshold 2 alert	*	
Threshold 1 alert	Threshold 1 alert	Only SW1 consults SW2 before alert
*	Threshold 2 alert	

(* = either Normal situation, Threshold 1 alert or Threshold 2 alert)

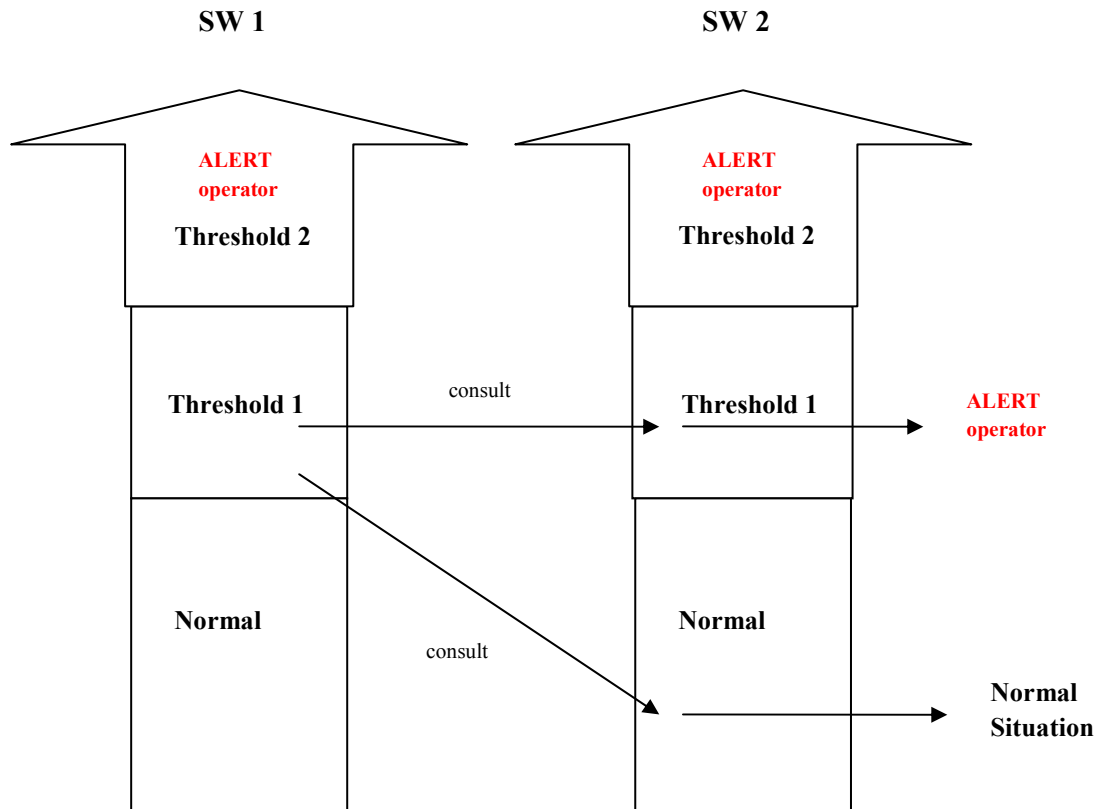


Figure 33: Different sliding window alert cases (No size or threshold value set, just for illustration)

5.3.2.2 Executing the Analyzer

The Analyzer starts by reading two configuration files located in the `configuration` folder at startup and configure the program accordingly. The program reads from the `settings.cfg` file the port to open for the listening socket and the threshold and size values for sliding windows 1 and 2, as shown in Figure 38. Since at this stage of the Ericsson IPsec implementation no statistics were available, therefore the setting of the appropriate variable values should be determined in future work. In the same folder the `boards.cfg` file is located and the Analyzer reads the number of boards to analyze, their IP addresses, and other data needed. The Analyzer also saves logs of the error packets in the `Log` folder.

5.3.3 Monitoring IPsec traffic

In order to update the sliding windows per SA, we need to keep track of the transmitted IPsec packets per SA. The idea is to mirror all the IPsec traffic to the Analyzer. The possible techniques are:

- Port-mirroring, also called SPAN (Switch Port Analyzer) in Cisco terminology: the switch replicates packets on a port for monitoring. See the reference for different switch manufacturers [75].

- TAP (Test Access Port): the tap sends traffic data to the monitoring device by splitting or regenerating the network signal, see [76]

Due to port-mirroring limitations, e.g.. packet loss and switch resources impact, passive TAP is considered a better solution, as discussed in [77]. However, since a TAP is a passive device, it will not filter anything, and the Analyzer machine has to be physically connected to the TAP device. Since the Analyzer has to manage more than one board and the different boards will not be connected to the same network, a sensor per board could be used to filter the IPsec traffic out of all the traffic directed to the board, encapsulate the SA selectors in a UDP packet and sent it to the Analyzer. In this way the amount of traffic sent to the Analyzer is less than mirroring all the traffic for each board to the Analyzer: the only information needed is the Analyzer ID (1 byte) and the SA selectors, Destination address (4 byte) and SPI (4 byte), encapsulated in an UDP packet (see Table 8).

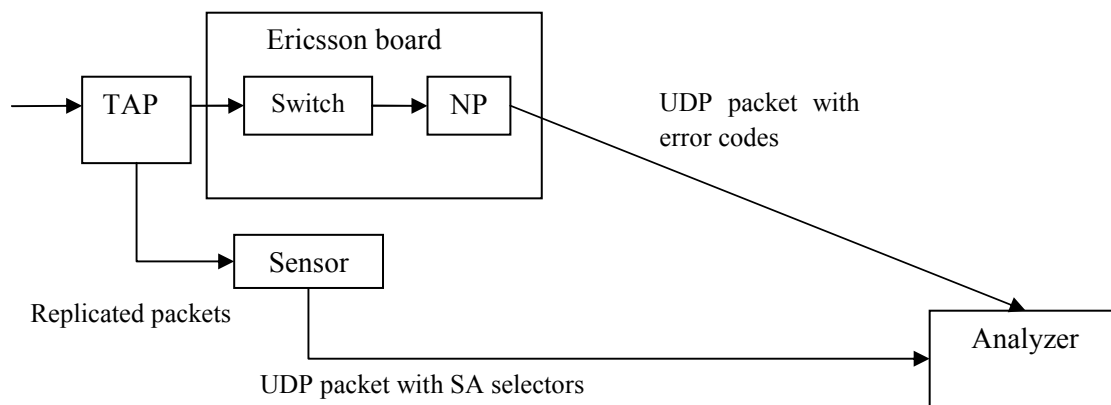


Figure 34: Configuration in case of data analysis from one Ericsson Ethernet Interface board, using TAP

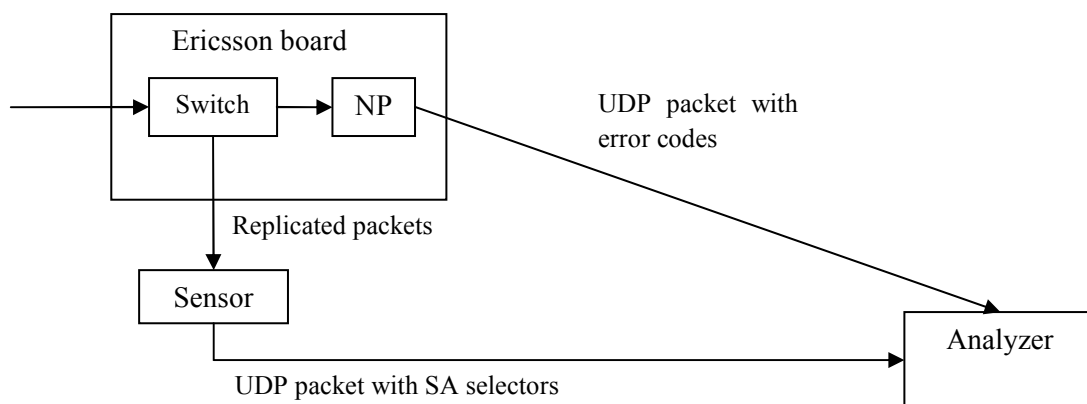


Figure 35: Configuration in case of data analysis from one Ericsson Ethernet Interface board, using port mirroring

5.3.4 *The Sniffer*

In order to monitor incoming IPsec traffic and emulate the TAP and sensor, we modified the example code of `sniffex.c` Version 0.1.1 (Copyright (c) 2005 The Tcpdump Group), found at [78]. The program captures data on the network using `libpcap` library [79]. Our modified version reads the network traffic and sends the IPsec SA selectors (SPI and Destination) as UDP packets to the Analyzer as specified in Table 8

6 ANALYSIS

6.1 Simulation in the LSI System Performance Analyzer

6.1.1 Capturing IPsec packets

To be able to modify IPsec packets for the simulation, network traffic containing IPsec traffic is dumped to a file through command line instructions, see Appendix A. The IPsec packet produced in the IPsec gateway simulation as described in Section 5.2 is used for later steps.

6.1.2 Reinjecting modified IPsec packets

Once the IPsec packet has been captured in the earlier steps, specific bits or field values can be modified. ESP SPI value, sequence number, and/or encrypted payload can be modified, which will generate an error when processed by the IPsec block. To generate error packets we ran some tests (see Appendix B), but we were not able to generate all the possible non-fatal errors, e.g. authentication failure, basic hash, TTL/HOP-limit underflow (see Table 3), because of limitations in the SPA. Once the packet has been modified, the IPsec payload (ESP header + payload + ESP trailer) is reinjected into the simulator by specifying the IP payload in the traffic generator as shown in Figure 36. The IP header is constructed as follow: the `Protocol` value is set to 50 (ESP) to indicate inbound processing and the `Destination IP address` value is set to select the correct IPsec protocol and mode. In a real implementation, the selectors of the IP packet will be used to lookup the policy in the SPD.

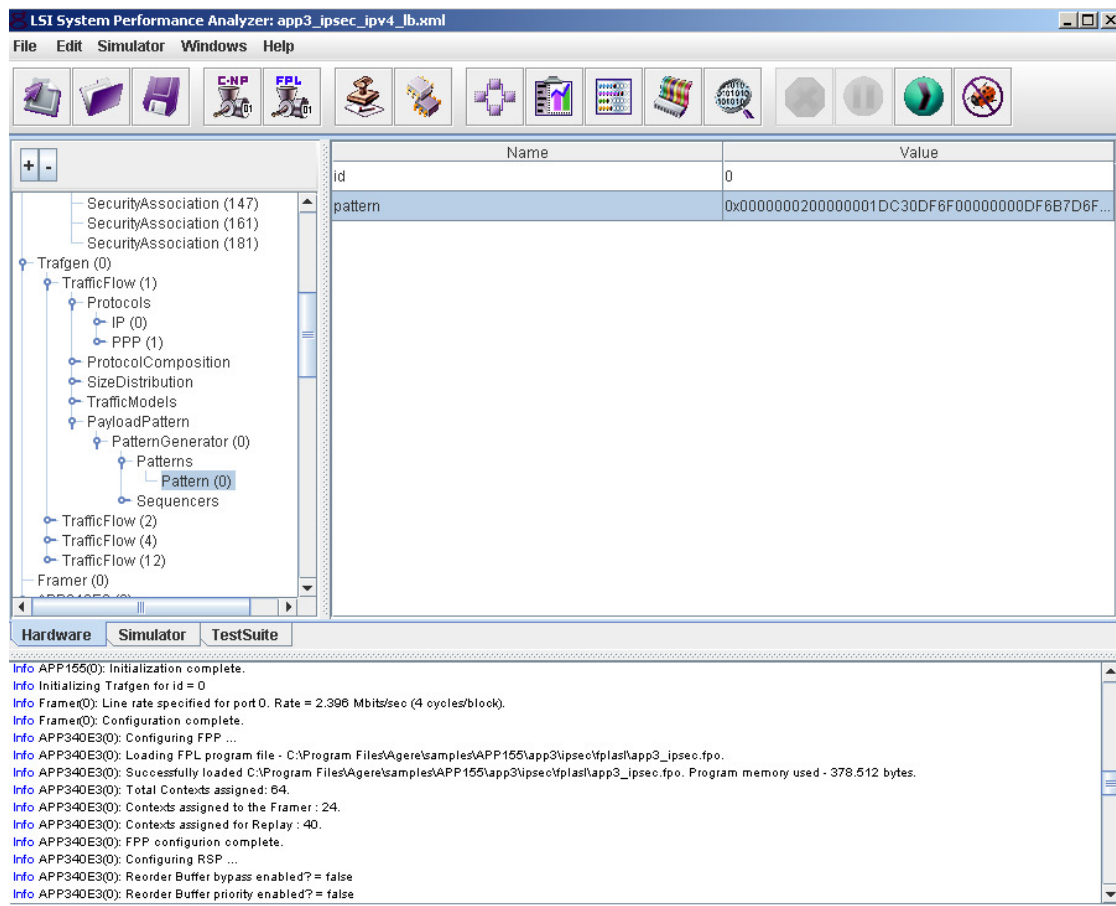


Figure 36: Screenshot of IPsec reinjection in the SPA simulator

6.1.3 Forwarding modified IPsec packets to the Analyzer

Once the modified IPsec packet has been reinjected in the simulator, the network processor processes the incoming packet as usual. When the modified packet is sent to the IPsec block for IPsec processing, a non-fatal error will occur during processing and the prepended SPP result header will indicate this error, see Figure 37. The IPsec block sends the error codes and IPsec packet to the network processor where and the CE assign it a DID (the DID created earlier in the simulator and specified in the CE FPL code). The DID selected (1" in this case) will forward the packet to the correct port by encapsulating the data into an UDP packet after executing the SED script described in Section 5.2.2.

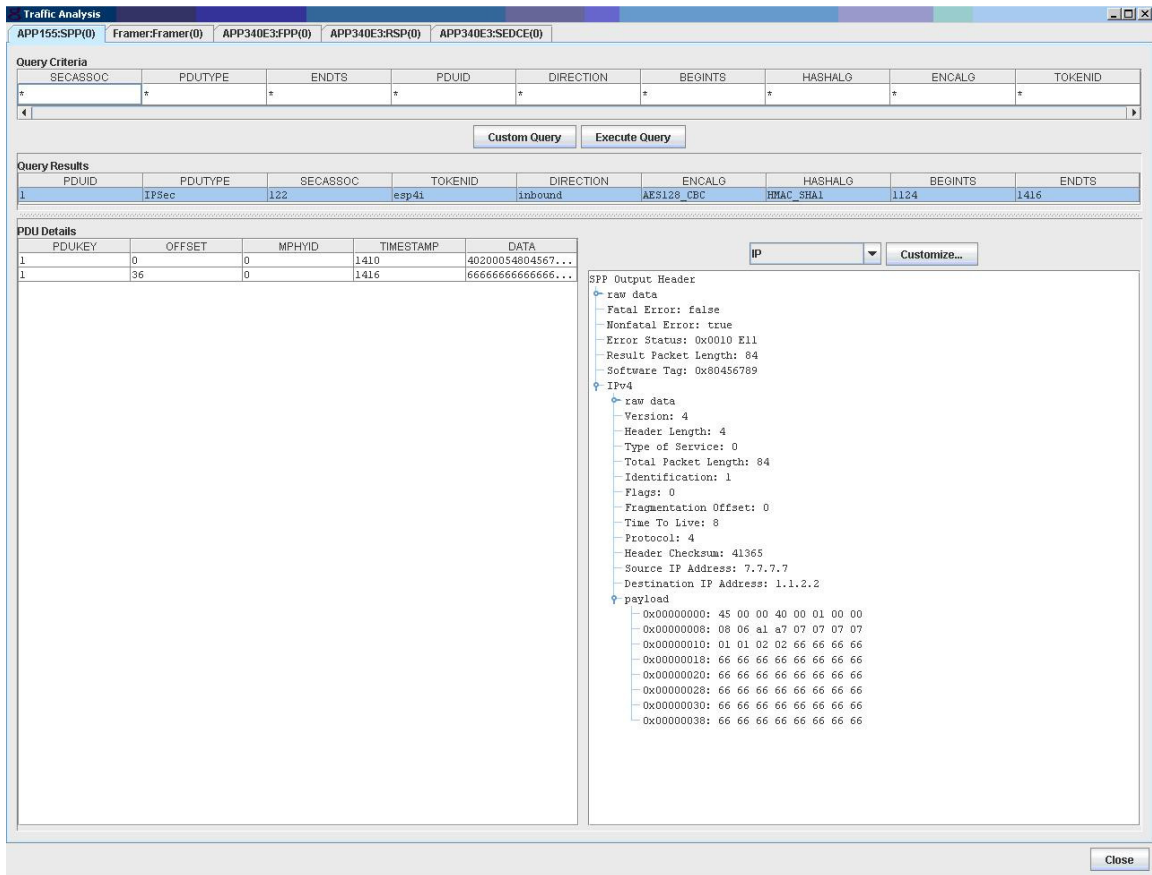


Figure 37: Screenshot of a modified IPsec packet with an IPsec block error set in the SPP result header in the SPA

6.2 Simulation using a software IPsec stack implementation

The purpose of the simulation is to show the different classes of attack, the potential abilities of the attacker, and to provide a functional test of the Analyzer implementation. The simulation did not define any threshold values, since no statistics were provided and, without them, discussion about reasonable threshold values is not possible. Therefore, all the thresholds in the Analyzer were set to zero which will generate both threshold alerts (see Figure 33).

```
134.138.45.182 - PuTTY
PRINTING PROGRAM SETTINGS

Network port          1700
DETECTION SETTINGS:
e1_sw1_size          20
e1_sw1_thres_1       0
e1_sw1_thres_2       0
e1_sw2_size          60
e1_sw2_thres_1       0
e1_sw2_thres_2       0
e2_sw1_size          20
e2_sw1_thres_1       0
e2_sw1_thres_2       0
e2_sw2_size          60
e2_sw2_thres_1       0
e2_sw2_thres_2       0
e3_sw1_size          20
e3_sw1_thres_1       0
e3_sw1_thres_2       0
e3_sw2_size          60
e3_sw2_thres_1       3
e3_sw2_thres_2       0

READING BOARD SETTINGS FROM CONFIGURATION FILE "configuration/boards.cfg"

SETTING DATA FOR BOARD "board1"
Allocating Memory for E1 SW
Allocating Memory for E3 SW
Allocating Memory for SA array

SETTING DATA FOR BOARD "board2"
Allocating Memory for E1 SW
Allocating Memory for E3 SW
Allocating Memory for SA array

SETTING DATA FOR BOARD "board3"
Allocating Memory for E1 SW
Allocating Memory for E3 SW
Allocating Memory for SA array

SETTING DATA FOR BOARD "board4"
Allocating Memory for E1 SW
Allocating Memory for E3 SW
Allocating Memory for SA array

SETTING DATA FOR BOARD "board5"
Allocating Memory for E1 SW
Allocating Memory for E3 SW
Allocating Memory for SA array

READING BOARD SETTINGS COMPLETED SUCCESSFULLY

Listening on UDP port: 1700
Press CTRL-C to terminate
```

Figure 38: Screenshot of the startup of Analyzer

The simulation required the following components:

Fast IPsec modified

Fast IPsec [80] is an implementation of IPsec, based on KAME [81], that can use cryptographic hardware devices whenever possible to carry out cryptographic operations, and consequently, to optimize the performance of IPsec. The Fast IPsec source code for NetBSD was modified to print a tag (ANALYZER_E[1,2,3]) in the system log file, when an E1, E2, or E3 failure occurs

Script to filter the logfile

```
tail -n0 -f /var/log/messages | awk -f analyzer_case
```

The script reads the last line of the logfile (option **-n0** of *tail*) and sends a line on the pipe as soon as it is written to the file (option **-f** of *tail*). *Awk* receives the new line from the pipe and calls the *analyzer_case* script (Appendix E), where is specified in case the line contains ANALYZER_E[1,2,3], send the packet (see Table 8) using the traffic generator created earlier

Traffic generator

see Section 5.3.1

Sniffer

see Section 5.3.4

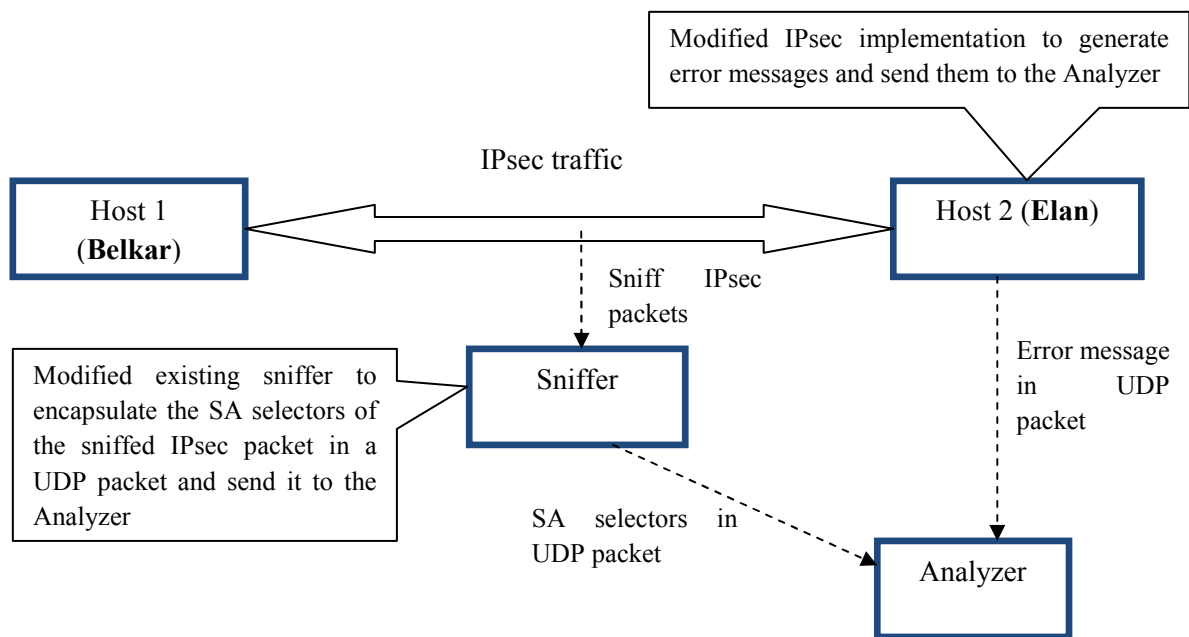


Figure 39: Logical configuration for simulation

The simulation was performed on two machines (NetBSD 3.0.2 with Fast IPsec enabled), named **Belkar** and **Elan**. The lab configuration and the programs executed on Belkar and Elan are shown in Figure 41. To set up the IPsec tunnel between Belkar and Elan, the SAD and SPD had to be configured on both machines. A good tutorial on how to set up an IPsec VPN can be found at [82]. The *setkey* command is used to set up the SAs, security policy and keys. The commands to be executed by setkey can be specified in a scriptfile. Both Belkar and Elan have their own key configuration files, called *setkey.conf*, which can be found in Appendix C. The *setkey.conf* script is executed by the following command under both machines:

```
/sbin/setkey -f /etc/setkey.conf
```

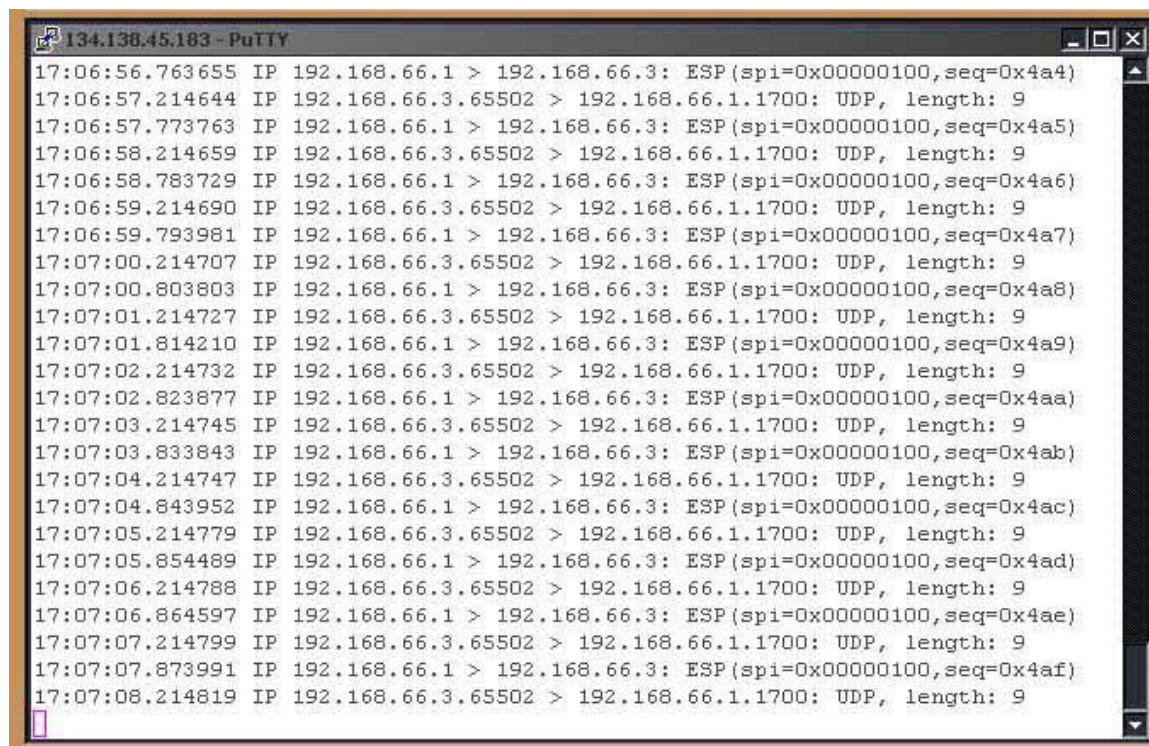
The content of the SAD can be shown executing the command:

```
/sbin/setkey -D
```

While the content of the SPD is dumped with:

```
/sbin/setkey -DP
```

After having configured the IPsec tunnel, we started the Analyzer, Sniffer, and the script to filter the logfile, monitoring the traffic with Tcpcmdump. To generate traffic from Belkar we executed the traffic generator. By monitoring the traffic with Tcpcmdump (version 3.8.3) we observed IPsec packets from Belkar to Elan, along with UDP packets (sniffer packets) from Elan to Belkar, as expected.



```
17:06:56.763655 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4a4)
17:06:57.214644 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:06:57.773763 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4a5)
17:06:58.214659 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:06:58.783729 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4a6)
17:06:59.214690 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:06:59.793981 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4a7)
17:07:00.214707 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:07:00.803803 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4a8)
17:07:01.214727 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:07:01.814210 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4a9)
17:07:02.214732 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:07:02.823877 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4aa)
17:07:03.214745 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:07:03.833843 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4ab)
17:07:04.214747 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:07:04.843952 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4ac)
17:07:05.214779 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:07:05.854489 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4ad)
17:07:06.214788 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:07:06.864597 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4ae)
17:07:07.214799 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
17:07:07.873991 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4af)
17:07:08.214819 IP 192.168.66.3.65502 > 192.168.66.1.1700: UDP, length: 9
```

Figure 40: Screenshot of Tcpcmdump trace of traffic between Belkar and Elan

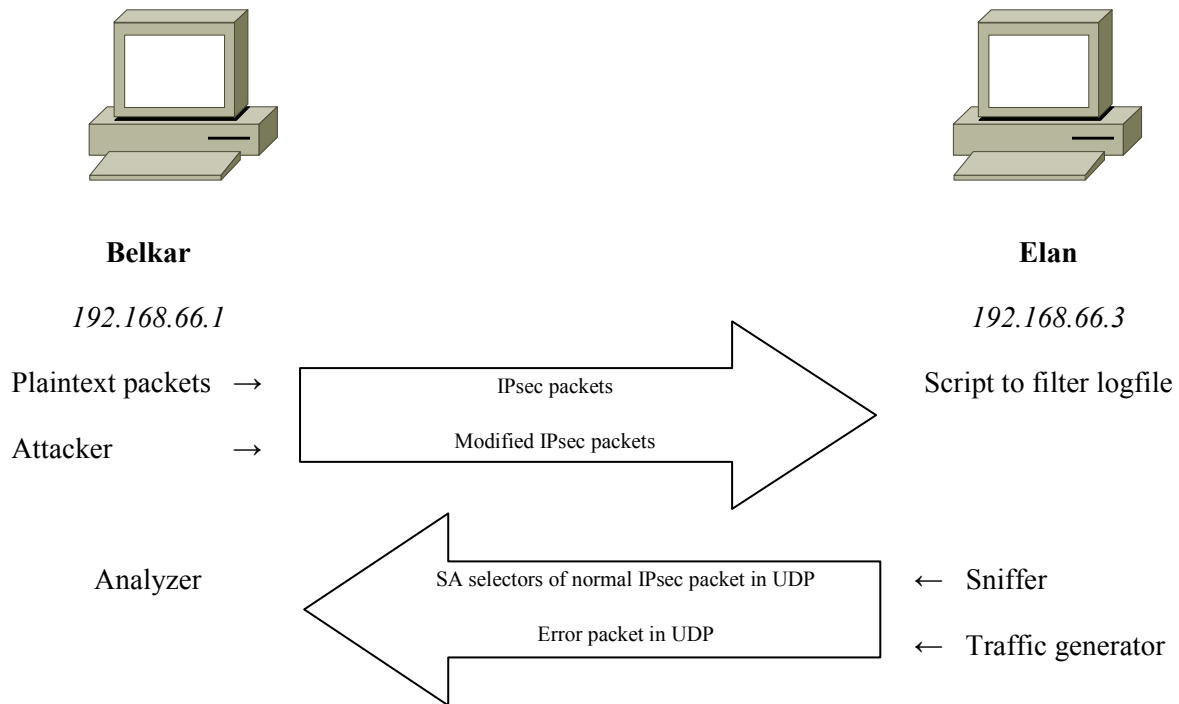


Figure 41: Configuration of the hosts in the lab and the flow of datagrams (traffic on interface wm0)

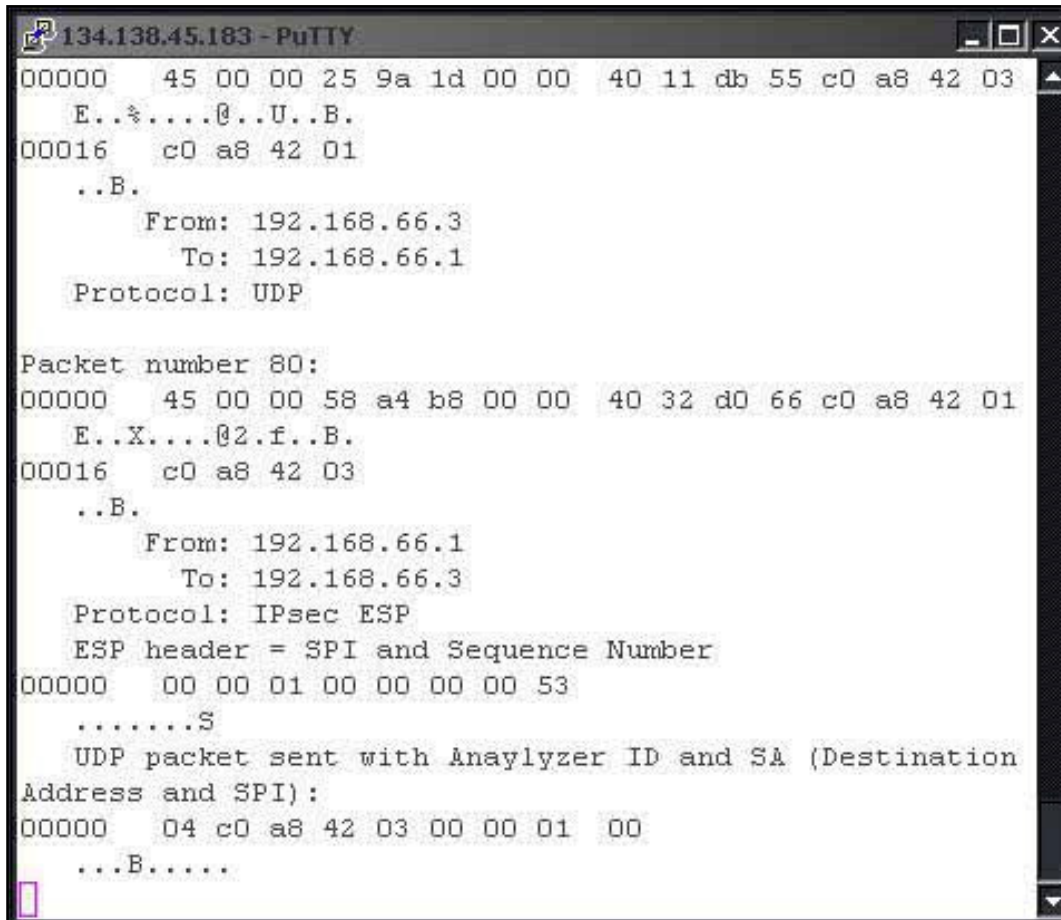
6.2.1 Simulation of attack at SA lookup phase

In this case the attacker has to forge a fake IPsec packet. It can be done using Netdude [83] (Network Dump data Displayer and Editor), a tool that allows a user to modify all fields of a packet stored in a pcap file. The attacker only needs to set the destination address of Elan, set the protocol field in the IP header to be 50 (ESP) and set the other fields of the packet. We forged a packet with Netdude (version 0.3.3-2.1), set SPI 0x666 and saved it in a pcap file SA_lookup_failure_SPI_666. To reinject the packet we used Tcpreplay (version 2.3.5) [84], using the command:

```
belkar# tcpreplay -I wm0 SA_lookup_failure_SPI_666
```

Once we reinjected this packet, we received the SA lookup alerts on the Analyzer, since the only allowed SPI was 256 (0x100), as specified in the *setkey.conf* for Elan, therefore the Analyzer printed two alerts since the threshold was set to 0 for both sliding windows.

6.2.1.1 Screenshots of programs running on Elan during SA lookup attack simulation



```
134.138.45.183 - PuTTY
00000  45 00 00 25 9a 1d 00 00  40 11 db 55 c0 a8 42 03
  E..%....@..U..B.
00016  c0 a8 42 01
  ..B.
      From: 192.168.66.3
      To: 192.168.66.1
      Protocol: UDP

Packet number 80:
00000  45 00 00 58 a4 b8 00 00  40 32 d0 66 c0 a8 42 01
  E..X....@2.f..B.
00016  c0 a8 42 03
  ..B.
      From: 192.168.66.1
      To: 192.168.66.3
      Protocol: IPsec ESP
      ESP header = SPI and Sequence Number
00000  00 00 01 00 00 00 00 53
  .....S
      UDP packet sent with Analyzer ID and SA (Destination
Address and SPI):
00000  04 c0 a8 42 03 00 00 01  00
  ...B.....
```

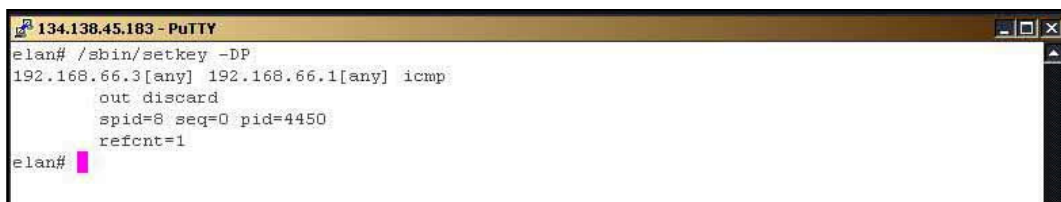
Figure 42: Screenshot of programs running on Elan during SA lookup attack simulation (Sniffer)



```
134.138.45.183 - PuTTY
elan# ./filter_logfile_cmd
Client is sending to 192.168.66.1:1700

Payload: 0x0145000058991c00004032dc02c0a84201c0a842030000006660000002c
```

Figure 43: Screenshot of programs running on Elan during SA lookup attack simulation (Script to filter logfile)



```
134.138.45.183 - PuTTY
elan# /sbin/setkey -DP
192.168.66.3[any] 192.168.66.1[any] icmp
out discard
spid=8 seq=0 pid=4450
refcnt=1
elan#
```

Figure 44: Screenshot of programs running on Elan during SA lookup attack simulation (SPD)

```

134.138.45.182 - PuTTY
12:24:28.150558 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4a)
12:24:28.365988 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:24:29.160568 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4b)
12:24:29.365945 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:24:30.170578 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4c)
12:24:30.365903 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:24:31.180596 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4d)
12:24:31.366003 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:24:32.190608 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4e)
12:24:32.365960 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:24:33.179443 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000666,seq=0x2c)
12:24:33.183783 IP 192.168.66.3.64687 > 192.168.66.1.1700: UDP, length: 29
12:24:33.200612 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x4f)
12:24:33.366060 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:24:33.367202 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:24:34.210669 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x50)
12:24:34.366304 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:24:34.944280 arp who-has 192.168.66.1 tell 192.168.66.3
12:24:34.944290 arp reply 192.168.66.1 is-at 00:0e:0c:b3:9f:32
12:24:35.220655 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x51)
12:24:35.365976 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:24:36.230670 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x52)
12:24:36.365933 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9

```

Figure 45: Screenshot of programs running on Elan during SA lookup attack simulation (Tcpdump)

6.2.1.2 Screenshots of programs running on Belkar during SA lookup attack simulation

```

134.138.45.182 - PuTTY

----- SA Lookup failure packet ALERT -----
board "board2"IP address      192.168.66.3
Received E1 packets (/SA)    1
SLIDING WINDOW 1:
Number error packets         1
Window size                   20
Threshold 1                   0
Threshold 2                   0
SLIDING WINDOW 2:
Number error packets         0
Window size                   60
Threshold 1                   0
Threshold 2                   0

----- SA Lookup failure packet ALERT -----
board "board2"IP address      192.168.66.3
Received E1 packets (/SA)    1
SLIDING WINDOW 1:
Number error packets         0
Window size                   20
Threshold 1                   0
Threshold 2                   0
SLIDING WINDOW 2:
Number error packets         1
Window size                   60
Threshold 1                   0
Threshold 2                   0

```

Figure 46: Screenshot of programs running on Belkar during SA lookup attack simulation (Analyzer)

```
134.138.45.182 - PuTTY
belkar# pwd
/home/ipsec0702/attacker
belkar# tcpreplay -i wm0 SA_lookup_failure_SPI_666
sending on: wm0
 1 packets (102 bytes) sent in 0.13 seconds
761194.0 bytes/sec 5.81 megabits/sec 7462 packets/sec
belkar# █
```

Figure 47: Screenshot of programs running on Belkar during SA lookup attack simulation (Attacker)

```
134.138.45.182 - PuTTY
belkar# /sbin/setkey -D
192.168.66.1 192.168.66.3
  esp mode=tunnel spi=256(0x00000100) reqid=0(0x00000000)
  E: 3des-cbc 7aeaca3f 87d060a1 2f4a4487 d5a5c335 5920fae6 9a96c831
  A: hmac-md5 c0291ff0 14dcccdd0 3874d9e8 e4cdf3e6
  seq=0x0000002c replay=0 flags=0x00000040 state=mature
  created: Feb 27 11:39:09 2008   current: Feb 27 12:22:49 2008
  diff: 2620(s)   hard: 0(s)     soft: 0(s)
  last:
  current: 0(bytes)   hard: 0(bytes)   soft: 0(bytes)
  allocated: 0   hard: 0   soft: 0
  sadb_seq=3 pid=29032 refcnt=2
belkar# /sbin/setkey -DP
192.168.66.1[any] 192.168.66.3[any] udp
  out ipsec
  esp/tunnel/192.168.66.1-192.168.66.3/require
  spid=20 seq=0 pid=20844
  refcnt=1
belkar# █
```

Figure 48: Screenshot of programs running on Belkar during SA lookup attack simulation (SAD and SPD)

6.2.2 Simulation of attack at IPsec processing phase

We captured some IPsec packets and stored them in a pcap file with the command:

```
belkar# tcpdump dst 192.168.66.3 -w capture
```

Then, we used Netdude to modify the (authenticated) payload of one of the sniffed packets, as shown in Figure 49.

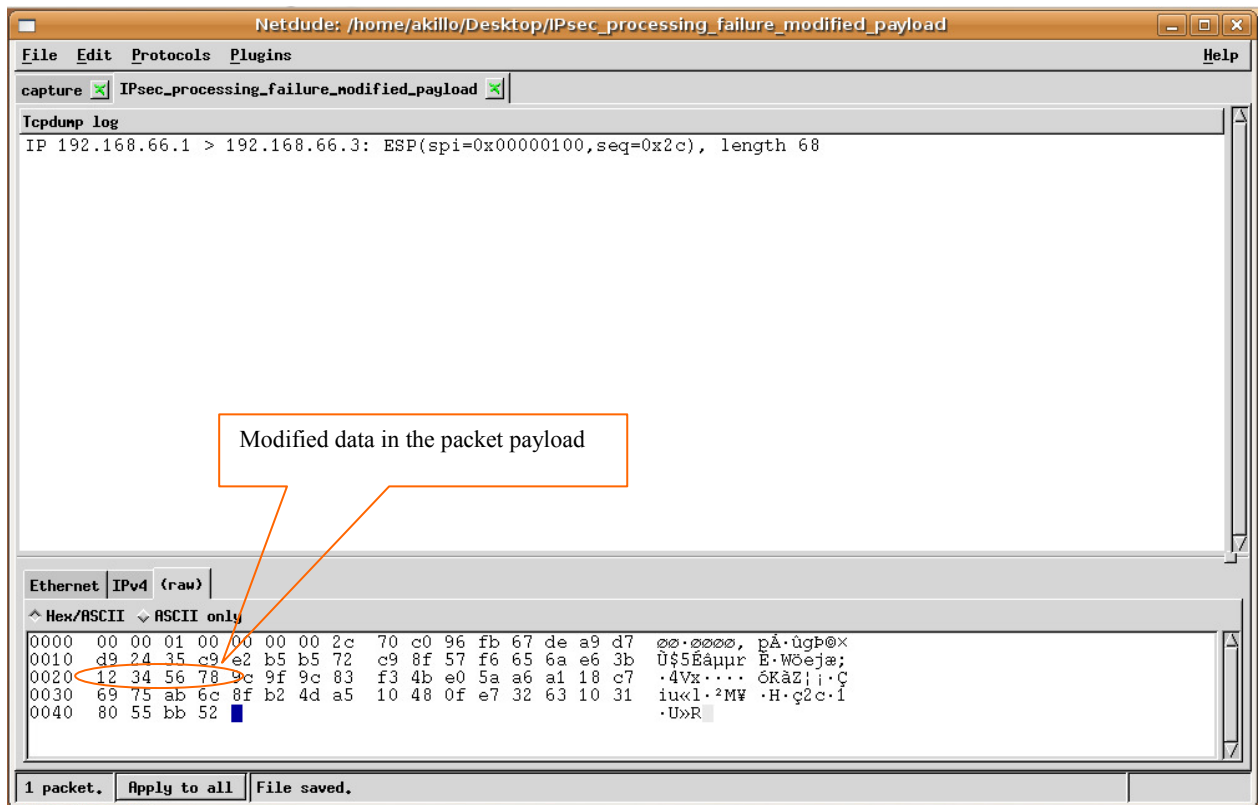


Figure 49: Screenshot of Netdude used for IPsec packet modification

We restarted Analyzer, Sniffer, and script to filter the log file. Then we used Tcreplay to reinject the packet:

```
belkar# tcreplay -I wm0 IPsec_processing_failure_modified_payload
```

This generated an IPsec processing failure alert on the Analyzer, since the packet failed the authentication validation (see Section 6.2.2.1 and 6.2.2.2).

6.2.2.1 Screenshots of programs running on Elan during IPsec processing failure

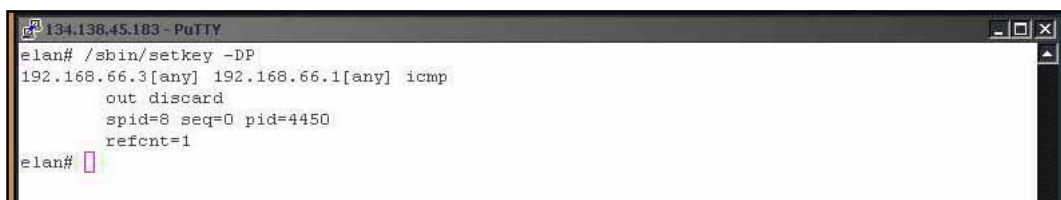


Figure 50: Screenshot of programs running on Elan during IPsec processing failure (SPD)

```

134.138.45.183 - PuTTY
elan# ./filter_logfile_cmd
Client is sending to 192.168.66.1:1700

Payload: 0x02402000548045678945000058991c00004032dc02c0a84201c0a842030000010
00000002c

```

Figure 51: Screenshot of programs running on Elan during IPsec processing failure (Script to filter logfile)

```

134.138.45.183 - PuTTY
00000 45 00 00 25 ca d5 00 00 40 11 aa 9d c0 a8 42 03
E..%....@.....B.
00016 c0 a8 42 01
..B.
From: 192.168.66.3
To: 192.168.66.1
Protocol: UDP

Packet number 1459:
00000 45 00 00 58 ca e6 00 00 40 32 aa 38 c0 a8 42 01
E..X....@2.8..B.
00016 c0 a8 42 03
..B.
From: 192.168.66.1
To: 192.168.66.3
Protocol: IPsec ESP
ESP header = SPI and Sequence Number
00000 00 00 01 00 00 00 02 76
.....V
UDP packet sent with Anaylyzer ID and SA (Destination
Address and SPI):
00000 04 c0 a8 42 03 00 00 01 00
...B.....

```

Figure 52: Screenshot of programs running on Elan during IPsec processing failure (Sniffer)

```

134.138.45.182 - PuTTY
12:33:39.618829 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x26c)
12:33:40.364699 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:33:40.628840 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x26d)
12:33:41.364800 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:33:41.638860 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x26e)
12:33:42.364614 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:33:42.648873 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x26f)
12:33:43.364714 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:33:43.658983 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x270)
12:33:44.364672 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:33:44.368905 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x271)
12:33:45.287377 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2c)
12:33:45.291918 IP 192.168.66.3.64685 > 192.168.66.1.1700: UDP, length: 37
12:33:45.364771 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:33:45.365484 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:33:45.678908 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x272)
12:33:46.365017 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:33:46.689200 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x273)
12:33:47.364688 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:33:47.698950 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x274)
12:33:48.364646 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:33:48.709002 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x275)
12:33:49.364745 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9

```

Figure 53: Screenshot of programs running on Elan during IPsec processing failure (Tcpdump)

6.2.2.2 Screenshots of programs running on Belkar during IPsec processing failure

```

134.138.45.182 - PuTTY
belkar# /sbin/setkey -D
192.168.66.1 192.168.66.3
    esp mode=tunnel spi=256(0x00000100) reqid=0(0x00000000)
    E: 3des-cbc 7aeaca3f 87d060a1 2f4a4487 d5a5c335 5920fae6 9a96c831
    A: hmac-md5 c0291ff0 14dccdd0 3874d9e8 e4cdf3e6
    seq=0x0000000c replay=0 flags=0x00000040 state=mature
    created: Feb 27 11:39:09 2008      current: Feb 27 12:22:49 2008
    diff: 2620(s)      hard: 0(s)      soft: 0(s)
    last:
    current: 0(bytes)      hard: 0(bytes)      soft: 0(bytes)
    allocated: 0      hard: 0      soft: 0
    sadb_seq=3 pid=29032 refcnt=2
belkar# /sbin/setkey -DP
192.168.66.1[any] 192.168.66.3[any] udp
    out ipsec
    esp/tunnel/192.168.66.1-192.168.66.3/require
    spid=20 seq=0 pid=20844
    refcnt=1
belkar#

```

Figure 54: Screenshot of programs running on Belkar during IPsec processing failure (SPD and SAD)

```

134.138.45.182 - PuTTY
----- IPsec processing failure packet ALERT -----
board "board2"IP address      192.168.66.3
Received E2 packets (/SA)    1
SLIDING WINDOW 1:
Number error packets         1
Window size                   20
Threshold 1                   0
Threshold 2                   0
SLIDING WINDOW 2:
Number error packets         0
Window size                   60
Threshold 1                   0
Threshold 2                   0

----- IPsec processing failure packet ALERT -----
board "board2"IP address      192.168.66.3
Received E2 packets (/SA)    1
SLIDING WINDOW 1:
Number error packets         0
Window size                   20
Threshold 1                   0
Threshold 2                   0
SLIDING WINDOW 2:
Number error packets         1
Window size                   60
Threshold 1                   0
Threshold 2                   0

```

Figure 55: Screenshot of programs running on Belkar during IPsec processing failure (Analyzer)

```

134.138.45.182 - PuTTY
belkar# pwd
/home/ipsec0702/attacker
belkar# tcpreplay -i wm0
IPsec_processing_failure_modified_payload
SA_lookup_failure_SPI_666
capture
belkar# tcpreplay -i wm0 IPsec_processing_failure_modified_payload
sending on: wm0
 1 packets (102 bytes) sent in 0.17 seconds
569832.4 bytes/sec 4.35 megabits/sec 5586 packets/sec
belkar# █

```

Figure 56: Screenshot of programs running on Belkar during IPsec processing failure (Attacker)

6.2.3 Simulation of attack at Security Policy verification phase

Before simulating this attack we had to change the security policy to discard the incoming traffic. We first tried with non-IPsec traffic, by flushing the SPD on Belkar and Elan, and pinging from Belkar to Elan. As it is shown in Figure 57 and Figure 58, we received an ICMP Reply for each ICMP Request (marked in dotted line). Once we set up the security policy to discard traffic going out from Elan, we did not receive ICMP Reply packets anymore (marked in full line).

```

134.138.45.183 - PuTTY
elan# /sbin/setkey -DP
No SPD entries.
elan# /sbin/setkey -f /etc/setkey.conf
elan# /sbin/setkey -DP
192.168.66.1[any] 192.168.66.3[any] any
    in discard
    spid=5 seq=0 pid=28595
    refcnt=1
elan#

```

Figure 57: Screenshot of Tcpcmdump trace of the traffic and SPD dump on Elan testing security policy enforcement (SPD)

```

134.138.45.182 - PuTTY
11:31:16.962869 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 380
11:31:17.963837 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 381
11:31:17.964111 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 381
11:31:18.952692 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 382
11:31:18.952927 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 382
11:31:19.962697 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 383
11:31:19.962884 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 383
11:31:20.962723 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 384
11:31:20.962842 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 384
11:31:21.962736 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 385
11:31:21.962943 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 385
11:31:22.962750 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 386
11:31:22.962901 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 386
11:31:23.962768 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 387
11:31:23.963000 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 387
11:31:24.962785 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 388
11:31:24.962958 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 388
11:31:25.962795 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 389
11:31:26.962824 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 390
11:31:27.962826 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 391
11:31:28.962844 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 392
11:31:29.962853 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 393
11:31:30.962872 IP 192.168.66.1 > 192.168.66.3: icmp 64: echo request seq 394

```

Figure 58: Screenshot of Tcpcmdump trace of the traffic and SPD dump on Elan testing security policy enforcement (Tcpcmdump)

We set up the security policy on Belkar to encrypt the outgoing ICMP Request packet (Figure 60) and setting the security policy on Elan to discard the incoming traffic as earlier. In this case we noticed something strange: we observed ICMP Reply packets (Figure 59) which means the packets received from Elan were IPsec processed, but no SPD lookup was actually done and no security policy was checked. We discovered subsequently that this is a bug in NetBSD Fast IPsec implementation [85].

```

134.138.45.182 - PuTTY
11:37:05.958200 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 729
11:37:06.958043 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x74)
11:37:06.958300 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 730
11:37:07.958059 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x75)
11:37:07.958258 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 731
11:37:08.958071 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x76)
11:37:08.958358 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 732
11:37:09.958086 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x77)
11:37:09.958315 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 733
11:37:10.958101 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x78)
11:37:10.958415 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 734
11:37:11.958121 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x79)
11:37:11.958373 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 735
11:37:12.958133 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x7a)
11:37:12.958473 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 736
11:37:13.958149 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x7b)
11:37:13.958431 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 737
11:37:14.958156 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x7c)
11:37:14.958388 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 738
11:37:15.958174 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x7d)
11:37:15.958489 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 739
11:37:16.958193 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x7e)
11:37:16.958308 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 740
11:37:17.958207 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x7f)
11:37:17.958405 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 741
11:37:18.958226 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x80)
11:37:18.958363 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 742
11:37:19.958241 IP 192.168.66.1 > 192.168.66.3: ESP (spi=0x00000100,seq=0x81)
11:37:19.958462 IP 192.168.66.3 > 192.168.66.1: icmp 64: echo reply seq 743

```

Figure 59: Tcpdump trace of the traffic on Belkar testing the faulty IPsec implementation on Elan

```

134.138.45.182 - PuTTY
belkar# /sbin/setkey -FP
belkar# /sbin/setkey -DP
No SPD entries.
sbin/setkey -f /etc/setkey.conf
belkar# /sbin/setkey -DP
192.168.66.1[any] 192.168.66.3[any] icmp
    out ipsec
    esp/tunnel/192.168.66.1-192.168.66.3/require
    spid=19 seq=0 pid=15567
    refcnt=2
belkar# ~

```

Figure 60: SPD dump on Belkar testing the faulty IPsec implementation on Elan

To go on with the simulation of this type of attack, we changed the security policy on Elan to discard the ICMP reply (outgoing traffic), and we restarted Analyzer, Sniffer and script to filter the log file. To have a Security Policy violation we started to ping from Belkar to Elan, and we obtained the alert messages on the Analyzer, as expected (see Section 6.2.3.1 and 6.2.3.2).

6.2.3.1 Screenshots of programs running on Elan during Security Policy violation

```

134.138.45.182 - PuTTY
12:35:28.364544 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:28.700476 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2d8)
12:35:29.364500 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:29.710518 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2d9)
12:35:30.364459 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:30.720497 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2da)
12:35:31.364417 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:31.730510 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2db)
12:35:32.364373 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:32.740526 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2dc)
12:35:33.364474 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:33.750540 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2dd)
12:35:34.364431 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:34.760555 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2de)
12:35:35.364388 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:35.770574 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2df)
12:35:36.364489 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:36.780683 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2e0)
12:35:37.364589 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:37.830595 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2e1)
12:35:38.364404 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9
12:35:38.840606 IP 192.168.66.1 > 192.168.66.3: ESP(spi=0x00000100,seq=0x2e2)
12:35:39.364505 IP 192.168.66.3.64688 > 192.168.66.1.1700: UDP, length: 9

```

Figure 61: Screenshot of programs running on Elan during Security Policy violation (Tcpdump)

```

134.138.45.183 - PuTTY
elan# /sbin/setkey -DP
192.168.66.3[any] 192.168.66.1[any] icmp
    out discard
    spid=8 seq=0 pid=4450
    refcnt=1
elan#

```

Figure 62: Screenshot of programs running on Elan during Security Policy violation (SPD)

```

134.138.45.183 - PuTTY
elan# ./filter_logfile_cmd
Client is sending to 192.168.66.1:1700

Payload: 0x0345000054d2270000ff01e622c0a84203c0a84201

```

Figure 63: Screenshot of programs running on Elan during Security Policy violation (Script to filter logfile)

```

134.138.45.183 - PuTTY
000000 45 00 00 25 d3 0b 00 00 40 11 a2 67 c0 a8 42 03
E...%....@..g..B.
00016  c0 a8 42 01
..B.
      From: 192.168.66.3
      To: 192.168.66.1
      Protocol: UDP

Packet number 1678:
000000 45 00 00 58 d0 b6 00 00 40 32 a4 68 c0 a8 42 01
E..X....@2.h..B.
00016  c0 a8 42 03
..B.
      From: 192.168.66.1
      To: 192.168.66.3
      Protocol: IPsec ESP
      ESP header = SPI and Sequence Number
000000 00 00 01 00 00 00 02 e2
.....
      UDP packet sent with Analyzer ID and SA (Destination
Address and SPI):
000000 04 c0 a8 42 03 00 00 01 00
...B.....

```

Figure 64: Screenshot of programs running on Elan during Security Policy violation (sniffer)

6.2.3.2 Screenshots of programs running on Belkar during Security Policy violation

```

134.138.45.182 - PuTTY
belkar# /sbin/setkey -D
192.168.66.1 192.168.66.3
esp mode=tunnel spi=256(0x00000100) reqid=0(0x00000000)
E: 3des-cbc 7aeaca3f 87d060a1 2f4a4487 d5a5c335 5920fae6 9a96c831
A: hmac-md5 c0291ff0 14dcccdd0 3874d9e8 e4cdf3e6
seq=0x0000002c replay=0 flags=0x00000040 state=mature
created: Feb 27 11:39:09 2008 current: Feb 27 12:22:49 2008
diff: 2620(s) hard: 0(s) soft: 0(s)
last: hard: 0(s) soft: 0(s)
current: 0(bytes) hard: 0(bytes) soft: 0(bytes)
allocated: 0 hard: 0 soft: 0
sadb_seq=3 pid=29032 refcnt=2
belkar# /sbin/setkey -DP
192.168.66.1[any] 192.168.66.3[any] udp
out ipsec
esp/tunnel/192.168.66.1-192.168.66.3/require
spid=20 seq=0 pid=20844
refcnt=1
belkar#

```

Figure 65: Screenshot of programs running on Belkar during Security Policy violation (SAD and SPD)

```
134.138.45.182 - PuTTY
----- IPsec policy violation packet ALERT -----
board "board2"IP address      192.168.66.3
Received E3 packets (/SA)    1
SLIDING WINDOW 1:
Number error packets         1
Window size                  20
Threshold 1                   0
Threshold 2                   0
SLIDING WINDOW 2:
Number error packets         0
Window size                  60
Threshold 1                   3
Threshold 2                   0

----- IPsec policy violation packet ALERT -----
board "board2"IP address      192.168.66.3
Received E3 packets (/SA)    1
SLIDING WINDOW 1:
Number error packets         0
Window size                  20
Threshold 1                   0
Threshold 2                   0
SLIDING WINDOW 2:
Number error packets         1
Window size                  60
Threshold 1                   3
Threshold 2                   0
```

Figure 66: Screenshot of programs running on Belkar during Security Policy violation (Analyzer)

```
134.138.45.182 - PuTTY
belkar# ping -i 120 192.168.66.3
PING 192.168.66.3 (192.168.66.3): 56 data bytes
```

Figure 67: Screenshot of programs running on Belkar during Security Policy violation (Attacker)

7 CONCLUSIONS

7.1 Conclusion

The aim of this thesis project was to implement an Analyzer prototype to identify anomalous activity on nodes when using an Ericsson Ethernet Interface board. We examined a variety of security weaknesses and attacks against IPsec protocol. A significant effort in this thesis was to understand the board, Network Processor, and its IPsec functionality.

We have met the goals stated for the thesis project, but we were not able to determine if the solution implemented is the best one. However, since the implementation is easily configurable this should facilitate future testing once the hardware is available. We have not been able to identify any weaknesses or attacks against IPsec ESP tunnel mode with authentication as used in this thesis but still detection of an attempt to attack the system is of relevance and reveals the compromised part of the network or incorrect implementation of IPsec. We tested different attack scenarios on an IPsec connection established between two hosts and the Analyzer detected all of the attacks successfully. The error threshold values were set to zero as explained earlier. We encountered several limitations and obstacles during our work, e.g. confidential information about the implementation of the IPsec coprocessor and the protocols used in the Ericsson Ethernet Interface board and the lack of a clear description of the non-fatal error codes. Other limitations included the incomplete implementation of IPsec both in hardware and software, which led us in splitting our effort into two parts: simulation of the board side and a prototype of the Analyzer. These limitations lead to the many assumptions that were made during this project. We had also to consider the requirements of executing scripts in the network processor; along with understanding the FPL and C-NP programming languages, as these had only a subset of the allowed functions typical of a high level programming language.

Through this thesis we gained a lot of understanding of the IPsec protocol, along with its weaknesses and different attacks against the protocol. We have also learned how the Ericsson Ethernet Interface board and its network processor functions e.g., packet classification, management, scheduling, etc. We have also learned how a network processor communicates with the IPsec block and how an IPsec packet is processed. We learned how to modify an IPsec implementation, setup an IPsec connection between two hosts, and reconfigure an operating system kernel. During the simulation phase we realized, after some testing, that a part of the NetBSD FAST IPsec implementation was incorrect and this was confirmed in an earlier bug report submitted (by others). This faulty IPsec implementation is exactly the same type of vulnerability exploited in Paterson's attack (described in Section 4.4.2).

Our suggestion for future thesis students is to carefully understand what you need to do, and have as detailed requirements as possible in the early stages of the work, even before starting the thesis project. Additionally, it is important also to keep the supervisors updated continuously, as in our case we wasted some time on an implementation that we subsequently had to redo.

7.2 Future work

There are many suggestions about what to do next. A suggestion would be to test the code which we have developed on the real hardware once the implementation is ready. That would require SAD, SPD, complete support of IPsec in the classification code, and SED scripts to process IPsec traffic. There is also a need to accumulate IPsec failure rate statistics in a simulated normal situation to be able to set the appropriate sliding window sizes and thresholds. Another suggestion for future work is to identify IPsec failure patterns and analyze IPsec packets by decrypting the payload. This would require synchronization with the SAD and SPD in order to synchronize keys and policies. A further area of research would be to investigate the possibility to prevent attacks by blocking suspicious IPsec traffic and investigate weaknesses in IPsec key management protocols.

REFERENCES

- [1] **3rd Generation Partnership Project**. *Technical Specification Group Services and System Aspects; 3G Security; Network Domain Security; IP network layer security*. 3GPP, December 2006. TS 33.210 V7.2.0.
- [2] **R. Atkinson and S. Kent**. *Security Architecture for the Internet Protocol*. IETF, November 1998. RFC 2401.
- [3] —. *IP Authentication Header*. IETF, November 1998. RFC 2402.
- [4] **R. Glenn and C. Madson**. *The Use of HMAC-MD5-96 within ESP and AH*. IETF, November 1998. RFC 2403.
- [5] —. *The Use of HMAC-SHA-1-96 within ESP and AH*. IETF, November 1998. RFC 2404.
- [6] **N. Doraswamy and C. Madson**. *The ESP DES-CBC Cipher Algorithms With Explicit IV*. IETF, November 1998. RFC 2405.
- [7] **R. Atkinson and S. Kent**. *IP Encapsulating Security Payload (ESP)*. IETF, November 1998. RFC 2406.
- [8] **D. Piper**. *The Internet IP Security Domain of Interpretation for ISAKMP*. IETF, November 1998. RFC 2407.
- [9] **D. Maughan, M. Schertler, M. Schneider and J. Turner**. *Internet Security Association and Key Management Protocol (ISAKMP)*. IETF, November 1998. RFC 2408.
- [10] **D. Carrel and D. Harkins**. *The Internet Key Exchange (IKE)*. IETF, November 1998. RFC 2409.
- [11] **R. Glenn and S. Kent**. *The NULL Encryption Algorithm and Its Use With IPsec*. IETF, November 1998. RFC 2410.
- [12] **R. Thayer, N. Doraswamy and Glenn R.** *IP Security Document Roadmap*. IETF, November 1998. RFC 2411.
- [13] **H. Orman**. *The OAKLEY Key Determination Protocol*. IETF, November 1998. RFC 2412.
- [14] **Erik Ahlform and Göran Örnulf**. Ericsson's family of carrier-class technologies. *Ericsson Review*. 2001, 04/2001, pp. 190-195.
- [15] **Victor Ferraro-Esparza, Michael Gudmandsen and Kristofer Olsson**. Ericsson Telecom Server Platform 4. *Ericsson Review*. 2002, 03/2002, pp. 104-113.
- [16] **Lars-Örjan Kling, Åke Lindholm, Lars Marklund and Nilsson Gunnar B.** CPP—Cello packet platform. *Ericsson Review*. 2002, 02/2002, pp. 68-75.
- [17] **LSI Corporation**. Network & Storage Standard Products. [Online] September 2007. http://www.lsi.com/documentation/networking/line_cards/NSPG_LineCard_082907.pdf. OT06-196OTH.

- [18] **N. Shah.** *Understanding Network Processors.* University of California. Berkeley, 2001. Master's Thesis.
- [19] **M. Kohler.** NP Complete. *Embedded Systems Programming.* November 2000, pp. 45-60.
- [20] **LSI Corporation.** Product Brief - APP300 - Access Network Processors. [Online] October 2006. http://www.lsi.com/documentation/networking/network_processors/LSI_PB_2pg_APP300.pdf. PB05-039NP-1.
- [21] **Agere Systems, Inc.** *APP300 Functional Programming Language (FPL) User Guide.* December 2004.
- [22] —. *APP300 Functional Programming Language (FPL) Reference Guide.* December 2004.
- [23] —. *APP300 C-NP Language Reference Guide.* December 2004.
- [24] PayloadPlus Product Briefs. *Agere Systems Inc.* [Online] 2003. [Cited: November 1, 2007.] <http://nps.agere.com/support/non-nda/index.htm#ProdBriefs>.
- [25] **Optical Internetworking Forum.** System Packet Interface Level 3: OC-48 System Interface for Physical and Link Layer Devices. [Online] June 2000. <http://www.oiforum.com/public/documents/OIF-SPI3-01.0.pdf>. IA#OIF-SPI3-01.0.
- [26] *Guide to Intrusion Detection and Prevention Systems (IDPS).* U.S. Department of Commerce, National Institute of Standards and Technology, February 2007. SP 800-94.
- [27] *Intrusion Detection Systems.* U.S. Department of Commerce, National Institute of Standards and Technology, November 2001. SP800-31.
- [28] **J. Postel.** *Internet Protocol.* IETF, September 1981. RFC 791.
- [29] **S. Deering and R. Hinden.** *Internet Protocol, Version 6 (IPv6) Specification.* IETF, December 1998. RFC 2460.
- [30] **F. Baker, D. Black, S. Blake and K. Nicolas.** *Definition of Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* IETF, December 1998. RFC 2474.
- [31] IANA Protocol Numbers. [Online] [Cited: October 8, 2007.] <http://www.iana.org/assignments/protocol-numbers>.
- [32] **Behrouz A. Forouzan.** *TCP/IP Protocol Suite.* 3rd Revised Edition. McGraw-Hill, 2005. p. 992. ISBN 0071115838.
- [33] *Guide to IPsec VPNs.* U.S. Department of Commerce, National Institute of Standards and Technology, January 2007. SP 800-77.
- [34] **Stephen J. Friedl.** An Illustrated Guide to IPsec. *Steve Friedl's Unixwiz.net Tech Tips.* [Online] August 24, 2005. [Cited: October 18, 2007.] <http://www.unixwiz.net/techtips/iguide-ipsec.html>.
- [35] **C. Kaufman, R. Perlman and M. Speciner.** *Network Security: Private Communication in a Public World.* 1. Prentice Hall, 1995. 0130614661.

- [36] **M. Bellare R. Canetti and H. Krawczyk.** *HMAC: Keyed-Hashing for Message Authentication.* IETF, February 1997. RFC 2104.
- [37] **R. Rivest.** *The MD5 Message-Digest Algorithm.* IETF, April 1992. RFC 1321.
- [38] *How to Break MD5 and Other Hash Functions.* **Wang Xiaoyun and Hongbo Yu.** 2005, EUROCRYPT.
- [39] *Finding collisions in the Full SHA-1.* **X. Wang, Y. L. Yin and H. Yo.** August 2005, Crypto 2005; Lectures Notes in Computer Science, Vol. 2621.
- [40] **R. Adams and R. Pereira.** *The ESP CBC-Mode Cipher Algorithms.* IETF, November 1998. RFC 2451.
- [41] *Data Encryption Standard (DES).* U.S. Department of Commerce, National Institute of Standards and Technology, October 1999. FIPS PUB 46-3.
- [42] **S. Kelly.** *Security Implications of Using the Data Encryption Standard (DES).* IETF, December 2006. RFC 4772.
- [43] **P. Karn, P. Metzger and W Simpson.** *The ESP Triple DES Transform.* IETF, September 1995. RFC 1851.
- [44] *Advanced Encryption Standard (AES).* U.S. Department of Commerce, National Institute of Standards and Technology, November 2001. FIPS PUB 197.
- [45] **Daniel J. Bernstein.** Cache-timing attacks on AES. [Online] November 11, 2004. <http://cr.yp.to/papers.html#cachetiming>.
- [46] **S. Frankel, R. Glenn and S. Kelly.** *The AES-CBC Cipher Algorithm and Its Use with IPsec.* IETF, september 2003. RFC 3602.
- [47] **R. Housely.** *Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP).* IETF, January 2004. RFC 3686.
- [48] MITRE Corporation. *CVE - Common Vulnerabilities and Exposures.* [Online] [Cited: 12 11, 2007.] <http://cve.mitre.org/>.
- [49] *VPNshield: Protecting CPN Services from Denial-of-Service (DoS) Attacks.* **M. Kaddoura, K. Millikin, R. Ramanujan, C. Sanders and J. Wu.** 2003. DARPA Information Survivability Conference and Exposition. Vol. II, p. 138.
- [50] Architecture Technology Corporation. *VPNshield: Protecting VPNs from DoS Attacks.* [Online] [Cited: 11 1, 2007.] <http://www.atcorp.com/secaresystems/vpnshield.html>.
- [51] **Mika Müller.** *Analysis Tool for Studying IP security Denial of Service Resistance .* Department of Computer Science, Helsinki University of Technology. March 2002. Master's Thesis.
- [52] **Ari Muittari.** *Internet Key Exchange (IKE) Protocol vulnerability risks.* Networking Laboratory, Electrical and Communication Engineering, Helsinki University of Technology. http://www.netlab.hut.fi/opetus/s38310/03-04/kalvot03-04/muittari_180504.ppt, 18 May 2004. Slides from a Master Thesis presentation.

- [53] **Matti Järvinen.** *PKI Requirements for IPsec*. Helsinki University of Technology. 2003.
- [54] **Jean Paul Degabriele and Kenneth G. Paterson.** Attacking the IPsec Standards in Encryption-only Configurations. *IEEE Symposium on Security and Privacy*. SP'07, 2007, 20-23 May 2007, pp. 335-349.
- [55] **Arnold K. L. Yau.** [Online] [Cited: March 3, 2008.] <http://ay2.org/ipsec/>.
- [56] **Henrik Dikvall.** *IPsec in Hardware*. Computer Science and Engineering, Luleå University of Technology. 2002.
- [57] **N. Ferguson and B. Schneier.** *A Cryptographic Evaluation of IPsec*. February 1999.
- [58] **Microsoft.** Step-by-step guide to Internet Protocol Security (IPsec). *Microsoft TechNet*. [Online] [Cited: March 3, 2008.] <http://technet.microsoft.com/en-us/library/bb742429.aspx#EGAA>.
- [59] *Probable Plaintext Cryptanalysis of the IP Security Protocols*. **S. M. Bellovin.** IEEE Computer Society, 1997, Proceedings of the 1997 Symposium on Network and Distributed System Security, p. 52. 0-8186-7767-8.
- [60] **G. Minshal.** Byte size distribution data. [Online] 1995. [Cited: november 1, 2007.] <http://www.nlanr.net/NA/Learn/Gm/pktsizes.html>.
- [61] **K. Claffy.** WAN packet size distribution. [Online] 1996. [Cited: November 1, 2007.] <http://www.nlanr.net/NA/Learn/packetsizes.html>.
- [62] **S. M. Bellovin.** *A Look Back at "Security Problems in the TCP/IP Protocol Suite"*. 2004.
- [63] **J.D. Touch and Y.E. Yang.** *Reducing the Impact of DoS Attacks on Endpoint IP Security*. IEEE, November 2006. 1424407745.
- [64] **K. G. Paterson and Arnold K. L. Yau.** *Cryptography in Theory and Practice: The Case of Encryption in IPsec*. [ed.] Vaudenay, S. Springer-Verlag, 2006. Vol. Lecture Notes in Computer Science Vol. 4004. 9783540345466.
- [65] **B. Liang and K. Muradyan.** *A Theoretical and Practical Overview of IPsec*. Department of Computer Science and Engineering, Chalmers University of Technology. Göteborg , 2006. Master's Thesis.
- [66] **S. M. Bellovin.** *Problem Areas for the IP Security Protocols*. San Jose, CA , July 1996.
- [67] Vulnerability Advisory 004033/NISCC/IPSEC. *UK, National Infrastructure Security Co-ordination Centre*. [Online] May 16, 2005. [Cited: 10 15, 2007.]
- [68] Vulnerability Note VU#302220. *US-CERT*. [Online] September 5, 2005. [Cited: October 18, 2007.] <http://www.kb.cert.org/vuls/id/302220>.
- [69] **Michael Barr.** CRC Mathematics and Theory. *Netrino*. [Online] <http://www.netrino.com/Embedded-Systems/How-To/CRC-Math-Theory>.
- [70] **Peter Wratil.** Transmitting safety-critical data over Industrial Ethernets. *The Industrial Ethernet Book*. [Online] <http://ethernet.industrial-networking.com/articles/articledisplay.asp?id=137>.
- [71] **Agere Systems, Inc.** *Simulator User's Guide*. November 2003.

- [72] —. *Security Protocol Processor Application Note*.
- [73] —. *APP300 Compute Engine Programming Guide*. December 2004.
- [74] **J. Postel**. *User Datagram Protocol*. IETF, August 1980. RFC 768.
- [75] Switch Reference. *Wireshark*. [Online] <http://wiki.wireshark.org/SwitchReference>.
- [76] **Datacom Systems, Inc.** [Online] <http://www.datacomsystems.com/solutions/choosing-network-taps.asp>.
- [77] **Tim O’Neill**. SPAN Port or TAP? CSO Beware. *LoveMyTool*. [Online] <http://www.lovemytool.com/blog/2007/08/span-ports-or-t.html>.
- [78] **The Tcpdump group**. *Tcpdump/Libpcap*. [Online] www.tcpdump.org/sniffex.c.
- [79] —. Pcap. *Tcpdump/Libpcap*. [Online] http://www.tcpdump.org/pcap3_man.html.
- [80] *Fast IPsec: A High-Performance IPsec Implementation*. **Samuel J. Leffler**. San Mateo, CA, USA : USENIX Association, September 8–12, 2003, Proceedings of BSDCon ’03.
- [81] *The KAME project*. [Online] <http://www.kame.net/>.
- [82] *The official IPsec Howto for Linux*. [Online] <http://www.ipsec-howto.org/>.
- [83] **Christian Kreibich**. *Netdude the Hacker's choice*. [Online] <http://netdude.sourceforge.net/>.
- [84] *Tcpreplay*. [Online] <http://tcpreplay.synfin.net/trac/>.
- [85] **Wolfgang Stukenbrock**. NetBSD-Bugs. [Online] [Cited: March 2, 2008.] <http://mail-index.netbsd.org/netbsd-bugs/2007/08/14/0000.html>.
- [86] **LSI Corporation**. Product Brief - APP100 - AAL2 SAR Co-Processor . [Online] June 2004. http://www.lsi.com/documentation/networking/network_processors/LSI_PB_4pg_APP.pdf. PB03-152NP.

APPENDIX A

SPA simulation traffic dump

```
np5.txt
Agere Dump APP300 Packets AG_NP-3.9.0.56(Wed
Aug 22 15:23:28 CDT 2007)

ASI Counters ...
512: 2
520: 2
524: 2
532: 2
540: 4
592: 1
608: 1
672: 1
676: 1

FPP Packets ...

[Packet #1]
(DestId:1022 Length:0 Offset:4 Params:0
TM Params:0 BeginTimestamp:391
EndTimestamp:2398)
[Segment #1]
Priority 0
Bytes Transmitted 64
Data FF 03 00 21 45 00 00 40
00 01 00 00 08 06 A1 A7
07 07 07 07 01 01 02 02
66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66

[Segment #2]
Priority 0
Bytes Transmitted 4
Data 66 66 66 66

[Packet #2]
(DestId:3002 Length:0 Offset:8 Params:0
TM Params:0 BeginTimestamp:4968
EndTimestamp:6971)
[Segment #1]
Priority 0
Bytes Transmitted 64
Data 00 00 00 88 00 45 67 89
45 00 00 88 00 00 00 00
0A 32 A1 36 06 06 06 06
01 01 02 02 00 00 00 02
00 00 00 01 DC 30 DF 6F
00 00 00 00 DF 6B 7D 6F
00 00 00 00 BA FF FF BF
FF FE FF FF F7 F9 5E 58

[Segment #2]
Priority 0
Bytes Transmitted 64
Data F8 F8 F8 F8 FE FE FD FD
99 99 99 99 99 99 99 99
99 99 99 99 99 99 99 99
99 99 99 99 99 99 99 99
99 99 99 99 99 99 99 99

[Packet #3]
(DestId:1122 Length:0 Offset:14 Params:0
TM Params:0 BeginTimestamp:8513
EndTimestamp:10568)
[Segment #1]
Priority 0
Bytes Transmitted 64
Data 05 05 05 05 05 05 09 09
09 09 09 09 08 00 45 00
00 88 00 00 00 00 0A 32
A1 36 06 06 06 06 01 01
02 02 00 00 00 02 00 00
00 01 DC 30 DF 6F 00 00
00 00 DF 6B 7D 6F 00 00
00 00 BA FF FF BF FF FE

[Segment #2]
Priority 0
Bytes Transmitted 64
Data FF FF F7 F9 5E 58 F8 F8
F8 F8 FE FE FD FD 99 99
99 99 99 99 99 99 99 99
99 99 99 99 99 99 99 99
99 99 99 99 99 99 99 99
99 99 99 99 99 99 99 99
99 99 FE FD FC FB FA F9

[Segment #3]
Priority 0
Bytes Transmitted 22
Data F8 F7 F6 F5 F4 F3 F2 F1
F1 FB 00 00 00 00 00 00
00 00 00 00 00 00

[Packet #4]
(DestId:3022 Length:0 Offset:0 Params:0
TM Params:0 BeginTimestamp:13091
EndTimestamp:14942)
[Segment #1]
Priority 0
Bytes Transmitted 64
Data 00 00 00 54 80 45 67 89
45 00 00 54 00 00 00 00
0A 04 A1 98 06 06 06 06
01 01 02 02 45 00 00 40
00 01 00 00 08 06 A1 A7
07 07 07 07 01 01 02 02
66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66

[Segment #2]
Priority 0
Bytes Transmitted 28
Data 66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66
```

```

66 66 66 66

RSP Packets ...

[Packet #1]
Destination ID      1022
Queue ID           1
Scheduler ID       0
Enqueue PDU ID    1
Packet Dropped?   No
Dropped Entity
Reason Dropped
Logical Port ID   4
Port Manager ID   4
Output Port ID    4
Traffic Manager Time 3 cycles
Traffic Shaper Time 1 cycles
Entry Timestamp   498
Exit Timestamp    3597
[Segment #1]
MPHY Port         0
Stream Editor Time 18 cycles
Timestamp         3567
Bytes Transmitted 64
Data             19 40 00 58 00 08 00 54
                00 45 67 89 14 00 10 00
                40 00 08 10 11 00 00 00
                45 00 00 88 00 00 00 00
                0A 32 A1 36 06 06 06 06
                01 01 02 02 45 00 00 40
                00 01 00 00 08 06 A1 A7
                07 07 07 07 01 01 02 02

[Segment #2]
MPHY Port         0
Stream Editor Time 18 cycles
Timestamp         3597
Bytes Transmitted 44
Data             66 66 66 66 66 66 66 66
                66 66 66 66 66 66 66 66
                66 66 66 66 66 66 66 66
                66 66 66 66 66 66 66 66
                66 66 66 66 66 66 66 66
                66 66 66 66

[Packet #2]
Destination ID      3002
Queue ID           3
Scheduler ID       0
Enqueue PDU ID    2
Packet Dropped?   No
Dropped Entity
Reason Dropped
Logical Port ID   3
Port Manager ID   3
Output Port ID    3
Traffic Manager Time 3 cycles
Traffic Shaper Time 1 cycles
Entry Timestamp   5077
Exit Timestamp    8185
[Segment #1]
MPHY Port         0
Stream Editor Time 1 cycles
Timestamp         8125
Bytes Transmitted 64
Data             05 05 05 05 05 05 09 09
                09 09 09 09 08 00 45 00
                00 88 00 00 00 00 0A 32
                A1 36 06 06 06 06 01 01
                02 02 00 00 00 02 00 00
                00 01 DC 30 DF 6F 00 00
                00 00 DF 6B 7D 6F 00 00
                00 00 BA FF FF BF FF FE

[Segment #2]
MPHY Port         0

```

```

Stream Editor Time 1 cycles
Timestamp         8168
Bytes Transmitted 64
Data             FF FF F7 F9 5E 58 F8 F8
                F8 F8 FE FE FD FD 99 99
                99 99 99 99 99 99 99 99
                99 99 99 99 99 99 99 99
                99 99 99 99 99 99 99 99
                99 99 FE FD FC FB FA F9

[Segment #3]
MPHY Port         0
Stream Editor Time 1 cycles
Timestamp         8185
Bytes Transmitted 22
Data             F8 F7 F6 F5 F4 F3 F2 F1
                F1 FB 00 00 00 00 00 00
                00 00 00 00 00 00

[Packet #3]
Destination ID      1122
Queue ID           1
Scheduler ID       0
Enqueue PDU ID    3
Packet Dropped?   No
Dropped Entity
Reason Dropped
Logical Port ID   4
Port Manager ID   4
Output Port ID    4
Traffic Manager Time 3 cycles
Traffic Shaper Time 1 cycles
Entry Timestamp   8620
Exit Timestamp    11805
[Segment #1]
MPHY Port         0
Stream Editor Time 16 cycles
Timestamp         11737
Bytes Transmitted 64
Data             1D 40 01 E8 00 98 00 88
                80 45 67 89 20 04 00 08
                42 00 10 00 50 00 00 08
                03 11 00 00 45 00 00 88
                00 00 00 00 0A 32 A1 36
                06 06 06 06 01 01 02 02
                00 00 00 02 00 00 00 01
                DC 30 DF 6F 00 00 00 00

[Segment #2]
MPHY Port         0
Stream Editor Time 16 cycles
Timestamp         11780
Bytes Transmitted 64
Data             DF 6B 7D 6F 00 00 00 00
                BA FF FF BF FF FE FF FF
                F7 F9 5E 58 F8 F8 F8 F8
                FE FE FD FD 99 99 99 99
                99 99 99 99 99 99 99 99
                99 99 99 99 99 99 99 99
                99 99 99 99 99 99 99 99
                99 99 99 99 99 99 99 99

[Segment #3]
MPHY Port         0
Stream Editor Time 16 cycles
Timestamp         11805
Bytes Transmitted 36
Data             99 99 99 99 99 99 99 99
                FE FD FC FB FA F9 F8 F7
                F6 F5 F4 F3 F2 F1 F1 FB
                00 00 00 00 00 00 00 00
                00 00 00 00

[Packet #4]
Destination ID      3022
Queue ID           4

```

```

Scheduler ID          0
Enqueue PDU ID       4
Packet Dropped?      No
Dropped Entity
Reason Dropped
Logical Port ID      2
Port Manager ID      2
Output Port ID       2
Traffic Manager Time 3 cycles
Traffic Shaper Time  1 cycles
Entry Timestamp      13198
Exit Timestamp       16228
[Segment #1]
MPHY Port            0
Stream Editor Time   16 cycles
Timestamp            16210
Bytes Transmitted    64
Data
05 05 05 05 05 05 09 09
09 09 09 09 08 00 45 00
00 40 00 01 00 00 08 06
A1 A7 07 07 07 07 01 01
02 02 66 66 66 66 66 66
66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66
66 66 66 66 66 66 66 66
[Segment #2]
MPHY Port            0
Stream Editor Time   16 cycles
Timestamp            16228
Bytes Transmitted    18
Data
66 66 66 66 66 66 66 66
66 66 66 66 66 66 9B DC
3C D9

```

testAPP155.txt

Agere Dump APP150 Packets AG_NP-3.9.0.56(Wed Aug 22 15:23:28 CDT 2007)

SE Packets ...

```

[Packet #1]
PDU Type              IPSec
Security Association ID 22
Token Template ID     esp4o
Direction             outbound
Encryption Algorithm   AES128_CBC
Authentication Algorithm HMAC_SHA1
Begin Timestamp       1086
End Timestamp          1394

[Packet #2]
PDU Type              IPSec
Security Association ID 122
Token Template ID     esp4i
Direction             inbound
Encryption Algorithm   AES128_CBC
Authentication Algorithm HMAC_SHA1
Begin Timestamp       3537
End Timestamp          3832

```


This generate an E11 error (SPI check failed) because the SPI 1 is not associated with Security Association 122.

Packet 3) SPI = 3 and Sequence number = 3

TrafficModels -> ConstantBitRate(0) -> duration = 3 packets

Payload pattern -> PatternGenerator(0) -> right click on Patterns -> Add new, Pattern -> Pattern(2) -> pattern = 0x0000000300000003(the rest as in test 1)

This generate an E11 error (SPI check failed) because the SPI number 3 is not defined in the SPA configuration file.

APPENDIX C

Setkey.conf scripts

Setkey.conf on Elan:

```
#!/sbin/setkey -f

# Key configuration script for Elan

# Flush the SAD and SPD
flush;
spdflush;

# ESP SAs doing encryption using 192 bit long keys (168 + 24 parity)
# and authentication using 128 bit long keys
add 192.168.66.1 192.168.66.3 esp 0x100 -m tunnel -E 3des-cbc
0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831 -A hmac-md5
0xc0291ff014dccdd03874d9e8e4cdf3e6;
add 192.168.66.3 192.168.66.1 esp 0x200 -m tunnel -E 3des-cbc
0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df -A hmac-md5
0x96358c90783bbfa3d7b196ceabe0536b;

# Security policies
spdadd 192.168.66.3 192.168.66.1 icmp -P out discard;
```

Setkey.conf on Belkar:

```
#!/sbin/setkey -f

# Key configuration script for Belkar

# Flush the SAD and SPD
flush;
spdflush;

# ESP SAs doing encryption using 192 bit long keys (168 + 24 parity)
# and authentication using 128 bit long keys
add 192.168.66.1 192.168.66.3 esp 0x100 -m tunnel -E 3des-cbc
0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831 -A hmac-md5
0xc0291ff014dccdd03874d9e8e4cdf3e6;
#add 192.168.66.3 192.168.66.1 esp 0x200 -m tunnel -E 3des-cbc
0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df -A hmac-md5
0x96358c90783bbfa3d7b196ceabe0536b;

# Security policies
spdadd 192.168.66.1 192.168.66.3 udp -P out ipsec esp/tunnel/192.168.66.1-
192.168.66.3/require;
```

APPENDIX D

SED Register File Map, Figure 2-11 from [73] (This figure appears here with the permission of LSI Corporation)

152	Reserved							
148		end_delta	begin_delta	frag_header				
144	frag_mtu		frag_offset					
143	Pre-Queue Parameters							
...								
128								
124	cre_start_offset	cre_end_offset	port_id					
120	start_offset	end_offset	logical_port					
116	pdu_length		pdu_offset					
112	prepend_bytes	flags	flags	flags				
108	ts_param_block							
96								
64	SED_param_block [64:95]							
0	data_block [0:63]							
	Slice A	Slice B	Slice C	Slice D				

APPENDIX E

Analyzer_case script

```
#!/bin/awk -f

/ ANALYZER_E1 / {
    # cut the first 28 byte; 1 byte = 2 characters -> first 56 char
    payload = substr($7, 0, 56)
    printf("Sending Analyzer E1 packet\n")
    printf("generator/tg -d 192.168.66.1 -p 1700 -P 0x01%s\n", payload) | "sh"
}

/ ANALYZER_E2 / {
    # cut the first 28 byte; 1 byte = 2 characters -> first 56 char
    payload = substr($7, 0, 56)
    # specify SPP header for authentication error
    printf("Sending Analyzer E2 packet\n")
    printf("generator/tg -d 192.168.66.1 -P 0x024020005480456789%s\n", payload)
| "sh"
}

/ ANALYZER_E3 / {
    # cut the first 20 byte; 1 byte = 2 characters -> first 40 char
    payload = substr($7, 0, 40)
    printf("Sending Analyzer E3 packet\n")
    printf("generator/tg -d 192.168.66.1 -P 0x03%s\n", payload) | "sh"
}
```