

Voice over IP and Lawful Intercept

Good cop/Bad cop

MUHAMMAD SARWAR JAHAN MORSHED



**KTH Information and
Communication Technology**

Master of Science Thesis
Stockholm, Sweden 2010

TRITA-ICT-EX-2010:28

Voice over IP and Lawful Intercept

God cop/Bad cop

Muhammad Sarwar Jahan Morshed

msjmo@kth.se

Masters Thesis

2010.02.20

This thesis is submitted in partial fulfilment of the requirements for a Masters of Science degree in Information & Communication Systems Security.

School of Information and Communication Technology
Royal Institute of Technology (KTH)
Stockholm, Sweden

Supervisor and Examiner: Professor Gerald Q. Maguire Jr.

Abstract

Lawful interception is a common practice for monitoring a telecommunication network by law enforcement agencies all over the world. It plays a vital role to ensure national security and to control crimes by providing authorized monitoring of communicating parties in a communication network. However, there are some important issues that need to be addressed, such as the privacy of individuals, malicious use of lawful interception by a “bad” cop, vulnerability of a lawful interception system to misuse by others, cost, legal liability, etc. These issues have led to opposition to lawful interception. Many researchers have been looking for a secure and acceptable lawful interception system that would eliminate or minimize the undesirable aspects of lawful interception. One of the approaches that gained a lot of attention is a key escrow encryption system. For lawful interception a key recovery key is escrowed with a trusted third party. This key can subsequently be used for decryption by the law enforcement agency. The trusted third party might be a government agency or a private company. The process for recovering keys should be based on a predefined security policy. The trusted third party’s responsibility is to store the key and to protect it from malicious use. This malicious use could be by a competitor, a telecommunication operator, Internet Service Provider (ISP), a law enforcement agency, or other party. If the trusted third party itself utilizes the key or improperly discloses the key to another party, then the data that was protected by encryption could be compromised. Unfortunately, there is no easy means to detect if the data has been tampered with or not. This thesis focuses on therefore in the case of voice over IP, where there is a need for a means to determine if a recorded conversation is authentic or not. Hence the objective of the overall thesis project is to design, implement, and evaluate a security mechanism that can be used with a trusted third party -based key escrow encryption system that will prevent or reduce the risk of forgery by (a bad cop within) a law enforcement agency using the escrowed key.

This thesis describes how a key escrow encryption system would be improved by the proposed mechanism – with a focus on the actions of a party that has access to the escrowed key. We do not examine how the party got access to this key, but for the purposes of this thesis we assumed that this party is either a good cop or a bad cop. We have defined the meaning of these terms and examine what operations a bad cop might attempt to perform – given the access to the master key. For example, this party could capture the data packets of a Voice over IP session, and then decrypt the packets using the key provided by the escrow agent. After decryption we examined the ability of a bad cop to modify or forge data packets, then encrypt these forged packets with the key – in order to fabricate evidence. We then examined how to detect such modifications or forgery. The proposed system is able to detect this forgery, based upon the inability of the forger to generate the correctly signed hashed message authentication code. We also examine additional extensions to the user agent and the escrow agent to be able to identify which packets (or groups of packets) were not generated by the original participant in the conversation. The goal is to understand if the proposed mechanism could make lawful interception more secure, while increasing the protection of the communicating parties’ conversation from undetected manipulation and making the digital record of a conversation easier to authenticate.

Key words: Lawful Intercept, VoIP, Key Escrow, Secure IP telephony, Law Enforcement Agency, Key Escrow Encryption System, Minisip, forgery detection, privacy, and bad cop.

Sammanfattning

Avlyssning är en vanlig metod för att övervaka ett telenät för brottsbekämpande organ i hela världen. Den spelar en viktig roll för att garantera nationell säkerhet och bekämpa brott genom att ge tillstånd övervakning av kommunikation parter i ett kommunikationsnät. Men det finns några viktiga frågor som behöver lösas, såsom den enskildes integritet, oönskad användning av avlyssning av en korrupt polis, sårbarheten hos en laglig avlyssning för att missbruk av andra kostnader, rättsliga ansvar, osv. Dessa frågor har lett till motstånd mot laglig avlyssning. Många forskare har letat efter en säker och acceptabel avlyssning system som skulle eliminera eller minimera oönskade effekter av laglig avlyssning. En av de strategier som fått mycket uppmärksamhet är en nyckeldeposition krypteringssystem. För avlyssning en nyckel återhämtning nyckel är escrowed med en betrodd utomstående. Denna nyckel kan därefter användas för dekryptering av de brottsbekämpande myndigheterna. Den betrodd utomstående kan vara en myndighet eller ett privat företag. Processen för att återställa nycklar bör grundas på en fördefinierad säkerhetspolitik. Den betrodda utomståendesansvar är att lagra nyckeln och att den skyddas från skadlig användning. Detta skadlig skulle kunna användas av en konkurrent, en teleoperatör, Internet Service Provider (ISP), en brottsbekämpande organ, eller annan. Om den betrodda utomstående själv använder nyckeln eller felaktigt lämnar ut nyckeln till annan, då de uppgifter som skyddas av kryptering kan äventyras. Tyvärr finns det inget enkelt sätt att upptäcka om data har manipulerats eller inte. Denna avhandling fokuserar på därför i händelse av Röst över IP, där det finns ett behov av ett medel för att avgöra om en inspelad konversation är giltig eller inte. Syftet med det övergripande examensarbete är att designa, implementera och utvärdera en säkerhet mekanism som kan användas med en betrodd utomstående-baserade nyckeldeposition krypteringssystem som kan förhindra eller minska risken för förfalskning av (en korruptmänniska inom) brottsbekämpande organ med escrowed nyckel.

Denna avhandling beskriver hur en nyckeldeposition krypteringssystem skulle kunna förbättras med den föreslagna mekanismen - med fokus på de åtgärder som en som har tillgång till escrowed nyckel. Vi undersöker inte hur partiet fick tillgång till denna nyckel, men inom ramen för denna avhandling vi utgått från att detta parti är antingen en laglydig polis eller en korrupt polis. Vi har definierat innebörden av dessa termer och undersöka vilka åtgärder som en korrupt polis kan försöka utföra - ges tillgång till huvudnyckeln. Till exempel kan detta parti fånga datapaketen i en Röstöver IP-session, och sedan dekryptera paket med hjälp av nyckeln som depositarien. Efter dekryptering vi undersökt möjligheterna för en elak polis att modifiera eller skapa datapaketen, sedan kryptera dessa förfalskade paket med nyckeln - för att fabricera bevis. Vi undersökte sedan hur man upptäcker sådana ändringar eller förfalskning. Det föreslagna systemet kan upptäcka denna förfalskning, baserat på oförmåga förfalskarens att generera korrekt undertecknade hashed meddelande autentisering kodad. Vi undersöker också ytterligare utvidgningar till användarprogram och depositarien att kunna identifiera vilka paket (eller grupper av paket) genererades inte av de ursprungliga deltagarna i samtalet. Målet är att förstå om den föreslagna mekanismen skulle kunna göra avlyssning säkrare, och samtidigt öka skyddet av kommunikation parternas samtal från oupptäckta manipulation och göra den digitala register över en omvandling lättare att verifiera.

Formatting Conventions

I have used the following paragraph styles for easy readability of this thesis paper:

- “Times New Romans” has been used for the body of the report.
- **An entity or component of a module is set in bold face.**
- *I have used italics to emphasize a paragraph or line or word.*

Table of Contents

Abstract	ii
Sammanfattning	iii
Formatting Conventions	iv
Table of Contents	v
List of Figures	ix
List of Tables	x
Acknowledgements	xi
List of Abbreviation and Acronyms	xii
Chapter 1: Introduction	1
1.1 Motivation.....	1
1.2 Synopsis of the Thesis	2
1.3 Research Problem	4
1.4 Research Methodology	4
1.5 Outline of the Thesis.....	5
Chapter 2: Background	6
2.1 Voice over Internet Protocol (VoIP).....	6
2.2 Session Initiation protocol (SIP).....	6
2.3 Lawful Intercept.....	7
2.3.1 General Concept of Lawful Intercept	7
2.3.2 Reason for Lawful Intercept	8
2.3.3 Basic Requirements for Lawful intercept	8
2.3.4 Ways of conducting lawful intercept	9
2.3.5 Lawful interception solutions	9
2.3.6 Existing rules and regulations for lawful intercept	9
2.3.7 Problems with lawful intercept	10
2.3.7.1 Privacy concerns	10
2.3.7.2 Vulnerabilities of (and due to) lawful interceptions	11
2.3.8 Lawful Interception Architecture.....	12
2.4 Trusted Third Party (TTP)	13
2.4.1 Definition of a Trusted Third Party	13
2.4.2 Requirements to be a Trusted Third Party	13
2.4.3 Public Key Infrastructure	14
2.4.4 Components of a PKI.....	15
2.4.5 Operation of a PKI.....	16
2.5 Digital Signature	16
2.6 Key Escrow	17
2.6.1 Key escrow encryption system	17
2.6.2 User Security Component.....	18

2.6.3	Key escrow component.....	19
2.6.4	Data recovery components.....	20
2.7	Secure Real Time Transport Protocol (SRTP).....	20
2.7.1	SRTP Architecture	20
2.7.2	SRTP Cryptographic Context (parameters and functions)	21
2.7.3	SRTP Algorithms.....	22
2.7.4	SRTP Procedure.....	23
2.7.5	Protection provided by SRTP	23
2.8	Secure Real Time Transport Control Protocol (SRTCP).....	23
2.9	Multimedia Internet KEYing (MIKEY)	24
2.9.1	General Concept of MIKEY	24
2.9.2	MIKEY Key Management Procedure.....	25
2.10	Key Agreement Schemes.....	26
2.10.1	Pre-Shared Key	26
2.10.2	Public Key Cryptography	26
2.10.3	Diffie-Hellman.....	27
2.10.4	DH-HMAC (HMAC authenticated Diffie- Hellman).....	28
2.10.5	RSA-R (Reverse RSA)	29
2.11	Minisip.....	29
2.12	Wireshark.....	30
Chapter 3: Related Work		31
3.1	C. Hett, et al.	31
3.2	Rafael Accorsi.....	31
3.3	V. Stathopoulos, et al.	32
3.4	Clipper Chip.....	32
Chapter 4: Design Analysis of the Proposed Model.....		33
4.1	Escrow Agent Module	33
4.1.1	Required Fields for Escrow Agent Module	33
4.1.2	Escrow Agent Database	34
4.1.3	User Agent Identification.....	36
4.1.4	Different URIs for User identification with the EA.....	36
4.1.5	Required parameters to escrow in future	37
4.1.6	Implementation Principles	37
4.2	LEA Module	38
4.2.1	Required parameters for the LEA module in order to provided the information required by the EA	38
4.2.2	Possible Trade-offs of the LEA Module.....	39
4.2.2.1	The time required to decode the recorded SRTP packets	39
4.2.2.2	Security of the LEA module	40
4.2.2.3	Network overhead.....	41
4.2.2.4	Transparency of the LEA module.....	41
4.2.3	Implementation principles of the LEA module	41
4.3	Attacker Module	42
4.4	Validation Module	42
4.5	Communication between UA and EA.....	42
4.6	Should the EA generate session keys for the LEA	43

Chapter 5: Implementation of the proposed LI model	44
5.1 User Agent	44
5.2 Escrow Agent (EA) module.....	44
5.3 LEA module.....	46
5.3.1 Algorithms of LEA module	47
5.3.2 Algorithm for calculating the ROC.....	47
5.3.3 Project Description.....	48
5.3.4 Capturing a Session.....	49
5.3.5 Operation procedure of the LEA module.....	49
5.4 Validation module.....	50
5.4.1 Algorithm for Validation module	50
5.4.2 Implementation of the Validation Module.....	51
5.4.3 Testing Forgery with the Attacker module	51
5.4.4 Algorithm for UDP Checksum Calculation.....	52
5.4.5 Working operation of the Attacker Module.....	53
5.4.6 Implementation of the attacker module	54
Chapter 6: Evaluation of the proposed LI System.....	56
6.1 Good cop Scenario	56
6.1.1 Time required for intercepting a session by the LEA	57
6.1.2 A Real-Life Example	61
6.2 Bad Cop Scenario	61
6.2.1 Possible ways of modifying a recorded call	62
6.2.2 Detection of the forgery	64
6.2.3 Shortcomings of the current Escrow Scenario.....	66
6.2.4 Overcoming this Limitation.....	67
6.2.5 Summary	67
Chapter 7: Future Work.....	68
Chapter 8: Conclusions.....	70
References.....	71
Appendices.....	75
A. Different Forgery Combination.....	75
B. Source code of LEA Module.....	84
C. Source code of Verification Module	94
D. Source code of Attacker Module.....	102
E. Required Time to derive Session Keys (SRTP packet/micro second)	126
F. Cumulative Percentage of the delays beyond the minimum required time to generate session keys.....	137
G. Required Time to decrypt SRTP packet (SRTP packet/micro second).....	138

H. Cumulative Percentage of the delays beyond the minimum required time to decrypt an SRTP packet.....	148
I. CPU used for performance Evaluation	149

List of Figures

Figure 1-1: Proposed Lawful Interception Model	3
Figure 2-1: Flow chart of a lawful interception (Adapted from ETSI 101 331[2]).....	7
Figure 2-2: HMAC (Adapted from [16][45])	12
Figure 2-3: Packet Cable Surveillance Model (Adapted from [15][16]).....	13
Figure 2-4: Digital Signature Workflow.....	16
Figure 2-5: Key Escrow Encryption System Components([16]).....	18
Figure 2-6: SRTP Packet Format (Adapted from[11])	21
Figure 2-7: Structure of a MIKEY Message (Adapted from [7])	26
Figure 2-8: Key Exchange using Public key Cryptography	27
Figure 2-9: Key Exchange using DH-HMAC	29
Figure 2-10: Key Exchange in RSA-R	29
Figure 5-1: LEA Module Login Interface.....	45
Figure 5-2: Interface for Providing Target Information	46
Figure 5-3: Escrow Information returned by the EA	46
Figure 5-4: Workflow of the Session Key Generation and Payload Decryption.....	50
Figure 5-5: Packets with Incorrect UDP Checksum are highlighted by Wireshark	52
Figure 5-6: A Packet Forged by Replacing the Content	55
Figure 5-7: Replaced Block by whole Content.....	55
Figure 6-1: Session Key Generation and Payload Decryption	57
Figure 6-2: Validation output of a Valid SRTP Session.....	57
Figure 6-3: Time required generating Session Keys from TGK and other escrowed information.....	58
Figure 6-4: Frequency vs. delay for Generating Session Keys.....	58
Figure 6-5: Cumulative frequency of the delay	59
Figure 6-6: Time required to decrypt an SRTP packet using session keys	59
Figure 6-7: Frequency vs. delay for decrypting SRTP packets	60
Figure 6-8: Cumulative frequency of the delay for decrypting SRTP packets.....	61
Figure 6-9: Forged block in the Front of the Session	65
Figure 6-10: Forged block in the Middle of the Session	65
Figure 6-11: Forged Block at the End of the Session	65
Figure 6-12: Visualized Form of a Forged Session	66

List of Tables

Table 1: Parameters that should be escrowed	34
Table 2: Used Database Tables	45
Table 3: All Possible Combinations of Forgery/modifications	63
Table 4: Examples of Forgery	64

Acknowledgements

I would like to start by thanking Professor Gerald Q. Maguire Jr., my thesis advisor and mentor for the last five months. I am grateful to him for his wonderful guidance, rapid support, encouragement, and as an endless sources of ideas. His breadth of knowledge and enthusiasm inspired me to carry out the thesis project successfully. I thank him for his valuable hours for discussing the thesis work with me, reading, verifying my work, and editing my writings. The research experience that I have had with him during this period, has given me confidence and encouragement for doing further research work in the future.

I want to thank Errik Eliasson for his support and valuable time. Additionally, I want to thanks my friends those who encourage me during this research work.

Finally, I would like to express my deep love to my wife Nurunnahar and my son Saanyaan for their continuous support and sacrifice during this thesis project. I also acknowledge to my parents for inspiring me to do a good thesis project that would serve as a base for my future career.

List of Abbreviation and Acronyms

AES	Advance Encryption System
AES-CM	Advance Encryption System in Counter Mode
ASP	Active Server Page
AVP	Audio/Video Profile
CA	Certificate Authority
CALEA	Communications Assistance for Law Enforcement Act
CC	Contents of Communication
CD	Call Data
CDR	Call Detail Records
CERT	Certificate
CFML	ColdFusion Markup Language
CODEC	Compression/decompression or Code/decode
CRL	Certificate Revocation List
CS	Crypto Session
CSBID	Crypto Session Bundle Identifier
CSID	Crypto Session Identifier
DH-HMAC	HMAC-based Diffie-Hellman
EA	Escrow Agent
ETSI	European Telecommunications Standard Institute
FISA	(U.S.)Foreign Intelligence Surveillance Act
GCKS	Group Controller/Key Serve
GPL	General Public License
HMAC	Keyed Hash-based Message Authentication Code
HMAC-SHA	Keyed Hash-based Message Authentication Code-Secure Hash Algorithm
HTTP	Hyper Text Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IIF	Internal Intercept Function
IP	Internet Protocol
ISP	Internet service provider
IV	Initial Vector
JSP	Java Server Page
LADP	Lightweight Directory Access Protocol
LEA	Law Enforcement Agency
LGPL	Lesser General Public License
LI	Lawful Interception
MAC	Message Authentication Code
MIKEY	Multimedia Internet KEYing
MIME	Multipurpose Internet Mail Extensions
MKI	Master Key Identifier
NSA	National Security Agency
NTP	Network Time Protocol
PEM	Privacy Enhanced Mail
PGP	Pretty Good Privacy
PHP	Hyphertext Preprocessor
PKI	Public Key Infrastructure

PRF	Pseudo Random Function
PSTN	Personal Switched Telephone Network
PTN	Public Telephone Network
RAND	Random
ROC	Roll Over Counter
RSA	Rivest Shamir Adleman
RSA-R	Reverse - RSA
RTCP	Real Time Transport Control Protocol
RTCP-XR	RTCP Extend Report
RTP	Real- Time transport Protocol
RTSP	Real-Time Streaming Protocol
SAP	Session Announcement Protocol
SAVP	Secure Audio Video Profile
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
SRTP	Secure Real-Time Transport Protocol
SSRC	Synchronization source
SRTCP	Secured Real Time Transport Control Protocol
TEK	Traffic Encryption Key
TGK	TEK Generation Key
TLS	Transport Layer Security
TTP	Trusted Third Party
UA	User Agent
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VoIP	Voice over Internet Protocol

Chapter 1: Introduction

The goal of this thesis project was to design and evaluate a Trusted Third Party based Lawful Interception (LI) system which would protect user's privacy from a bad cop within a Law Enforcement Agency (LEA), while serving the purpose of lawful interception to protect the homeland security as well as to investigate crime. This chapter provides the thesis overview, research problem, and the research methodology to carry out the thesis.

1.1 Motivation

LI is widely used by the LEAs for monitoring telecommunication traffic. It plays a vital role to ensure national security and to control crime by providing authorized monitoring of the communicating parties in a telephone (or other communication) system. While many people claim that intercepting of a conversation violates basic human rights and individual privacy, many national governments have passed laws and regulations enabling lawful interception. Because of the need to balance the interests of society and the rights of individuals there are some important issues that need to be addressed, such as the privacy of individuals, malicious use of lawful interception by a "bad" cop, vulnerability of a lawful interception system to misuse by others, cost, legal liability, etc. These issues have led to opposition to lawful interception. This has been discussed further in section 2.3.7.

Traditional telephony was introduced ~100 years ago and lawful interception of traditional telephony has been well established in both law and practice for many decades. However, with the introduction of Voice over IP (VoIP), users have rapidly switched from traditional telephony to VoIP because of its speed, support for mobility, and due to its digital nature encryption is easy to do – hence greater privacy can be implemented. Unfortunately, VoIP has also become popular for criminals – as it enables them to have a secure conversation, frustrating intercepting of VoIP conversations by the law enforcement agencies. The difficulties of interception are due to the VoIP architecture and the use of smart end devices – thus it is hard to intercept both the signaling and the call contents. For details of these problems see [15].

In recent years, a number of proposals have been presented by research and academic organizations to create a more secure and more acceptable lawful interception system. One of the approaches that has gained a lot of attention is a key escrow encryption system. For lawful interception a key recovery key is escrowed with a trusted third party. This key can subsequently be used for decryption by the law enforcement agency. The trusted third party might be a government agency or a private company. The process for recovering keys should be based on a predefined security policy. The trusted third party's responsibility is to store the key and to protect it from malicious use. This malicious use could be by a competitor, a telecommunication operator, Internet Service Provider (ISP), a law enforcement agency, or other party. If the trusted third party itself utilizes the key or improperly discloses the key to another party, then the data that was protected by encryption could be compromised. For example, if the key was disclosed to a "bad" cop, then this person would not only be able to decrypt the data, but as the encryption that is

generally used is symmetric encryption – they now have the ability to generate data and encrypt it with the key – thus forging evidence. Unfortunately, there is no easy means to detect if the data has been tampered with or not. Therefore in the case of voice over IP, there is a need for a means to determine if a recorded conversation is authentic or not.

1.2 Synopsis of the Thesis

Use of VoIP is increasing daily as it provides both additional services and lower costs than traditional telephony system. However because of the ease of encrypting the contents of a media session, the use of VoIP became an issue of concerning for law enforcement agencies in all countries. For this reason and concerns of competition with government owned telecommunications operations the use of VoIP is restricted in many countries. In a traditional telephony system the government can easily monitor the communicating parties who are using a telecommunication network as the network has centralized control and all of the communication passes through telephone exchanges that are controlled by the telecommunications operator. However, the architecture of a VoIP system makes it more difficult to intercept traffic because the call signaling and the communication between the parties may travel along completely different paths and need **not** even travel through the same network operators. Additionally, the end points of the communication are intelligent and capable of doing encryption/decryption, using a mutually agreed but **non-standard** CODEC, etc. All of these characteristics make successful interception much harder than the case of a traditional telephony network.

Additionally, lawful interception has become a contentious issue between law enforcement agencies and the citizens of the country. Many argue that lawful interception violates basic human rights as the individual's expectation of privacy may be substantially compromised. While at the same time lawful interception is a useful tool for ensuring national security as well as for investigating criminal activity that may lead to a criminal prosecution. Further compounding the problem is evidence that lawful interception may introduce security holes in communication systems, thus making it easier for criminals or terrorists to interrupt or manipulate the operation of a communication network for their own benefit. Therefore a secure lawful interception mechanism is highly desirable, but it should be reliable and acceptable to all (lawful) parties. Several proposals have been presented over the last three decades. Some of these proposals offer good prospects for increasing the reliability and security of lawful interception [4][5][47]. Key escrow encryption is one such proposal. In this approach the master key (also known as a key recovery key) will be escrowed with a trusted third party for future use. For example, a court might issue a warrant to the third party to disclose the escrowed key for a specific session (or sessions during a period of time) to enable a law enforcement agency to access the signaling or contents of this session or sessions. The trusted third party must be very reliable (i.e., they need to be able to provide the escrowed key when appropriate). Additionally, the trusted third party must be trusted to only provide the escrowed key when presented with a lawful order to do so; otherwise they must **not** provide a key to a law enforcement agency or any other party.

Figure 1-1 shows an overview of the process of escrowing a key and the subsequent recovery of a key for the purposed of lawful interception (LI) by a law enforcement agency (LEA) following presentation of a warrant to the trusted third party (TTP). We will examine the details of these interactions in the next chapter.

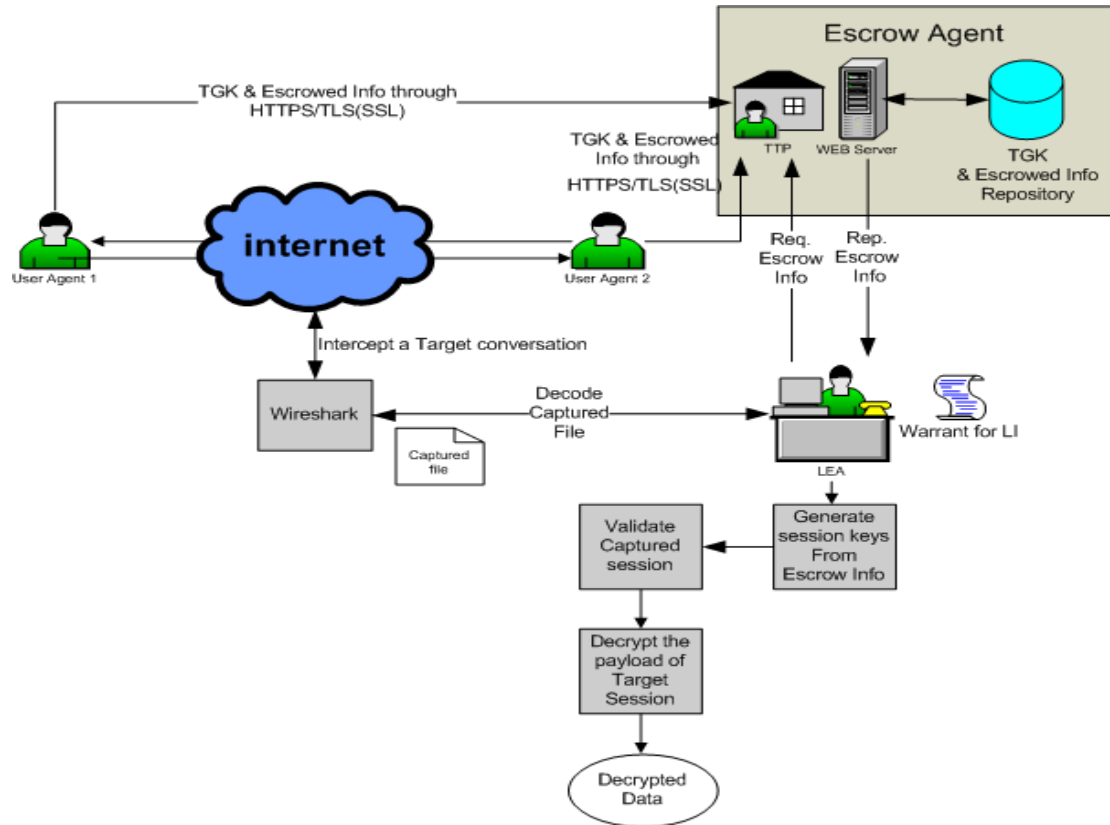


Figure 1-1: Proposed Lawful Interception Model

The objective of the overall thesis project is to design, implement, and evaluate a security mechanism that can be used with a trusted third party -based key escrow encryption system that will prevent or reduce the risk of forgery by (a bad cop within) a law enforcement agency using the escrowed key. As part of this project, Md. Sakhawat Hossen has designed and implemented the proposed security mechanism using an open source session initiation protocol user agent (minisip), for details see his thesis [43]; minisip was developed at the Royal Institute of Technology (KTH), Sweden [8]. Minisip supports both MIKEY and SRTP. In MIKEY, a master key is established by MIKEY to perform a key exchange. This master key is used to derive a key to be used by SRTP to protect a media stream and by SRTCP to protect the associated RTCP information. The modified version of this user agent escrows the session “Master Key” with a trusted third party. Additionally, Md. Sakhawat Hossen has extended the user agent’s MIKEY+SRTP security by utilizing a digital signature to sign Hashed Message Authentication Codes (HMACs) computed over the RTP and RTCP message contents.

This thesis describes how a key escrow encryption system would be improved by the proposed mechanism – with a focus on the actions of a party that has access to the escrowed key. We do not examine how the party got access to this key, but for the purposes of this thesis we will assume that this party is either a good cop or a bad cop. We will define the meaning of these terms and examine what operations a bad cop might attempt to perform – when given access to the master key. For example, this party could capture the data packet using Wireshark [19], a third party network analyzer, then decrypt the packets using the key provided by the escrow agent. After

decryption a bad cop might attempt to modify or forge data packets, then encrypt these forged packets with the user's session key – in order to fabricate evidence. We will examine how to detect such modifications or forgery. The proposed system should enable detection of this forgery, based upon the inability of the forger to generate the correctly signed HMACs. We will also examine additional extensions to the user agent and the escrow agent in order to be able to identify *which* packets (or groups of packets) were **not** generated by the original participant in the conversation. The goal is to understand if the proposed mechanism can make lawful interception more secure, while increasing the protection of the communicating parties' conversation from undetected manipulation and making the digital record of a conversion easier to authenticate.

1.3 Research Problem

Lawful interception is a useful tool for law enforcement in almost all countries to investigate crimes. In fixed and mobile telephony systems the signaling and media are carried over a network operated by a telecommunications operator over whom the government generally has a lot of control (in some cases the government may even operate this network). In many countries all such operators have to provide facilities for lawful interception as a requirement for offering telecommunication services [15][16]. In addition to the mandatory requirement to provide facilities for law enforcement agencies to conduct lawful intercepts, the laws in most countries provide clear descriptions of how lawful interception is to be carried out, who must pay (and when), and who is responsible for doing (or not doing) what.

In contrast, successful interception of VoIP communications is difficult as the packets exchanged between two (or more) communicating parties may follow one or **more** routes from the source to the destination over the Internet [15]. Additionally, the signaling to establish a session and the media traffic of a session may take completely different paths through the network. Hence the distributed nature of VoIP architecture versus the centralized nature of traditional telephony systems makes capturing the raw traffic difficult.

Moreover, many people and organizations do not support the use of lawful interception to protect national security or to control crime in a country. Rather they more concern about preserving the individual's privacy and consider lawful interception an abuse of human rights. To address these conflicting issues, many proposals have been presented [4][45]. A trusted third party based key escrow encryption system is a widely accepted solution in this regard. In this system a master key for each session is escrowed with a trusted third party, this key could be used for subsequent decryption by a law enforcement agency. However, there is limitation in this mechanism - as a bad cop within the law enforcement agency could decrypt the captured data, modify this data, and then encrypt the resulting data using the session key. Unfortunately, the communicating parties will be unable to prove themselves innocent in the court, since the modified data is indistinguishable from the original data. In this thesis, we will examine enhancements made to a key escrow encryption system to provide a solution for lawful interception of VoIP communication while avoiding the possibility for undetected modification or fabrication of session contents.

1.4 Research Methodology

The overall thesis project was conducted as two related projects. Both projects have built upon the existing minisip [8] open source session initiation protocol user

agent. The work was divided into two sub-projects: enhancements to the user agent (described in the thesis by Md. Sakhawat Hossen [43]) and attacks upon this system by a potentially "bad" cop (the topic of this thesis). Additionally, the later contributed to the former thesis by suggesting additional data that should be escrowed given the needs of a LEA. In more detail these two subprojects were:

1. Design, implementation, and evaluation of a modified Minsip that can escrow the session's master key at the end of a session with an escrow agent. This escrow agent will act as our trusted third party. This sub-project is the topic of Md. Sakhawat Hossen's thesis [43].
2. Further development of the escrow agent and the design, implementation, and evaluation of (a) a law enforcement module, (b) an attacker module, and (c) a validation module. The law enforcement module should enable a Law Enforcement Agency to decode a captured VoIP session using the master key provided by the escrow agent. The attacker module will modify a captured VoIP session. Finally, the validation module will be used to detect the modifications or fabrication of a VoIP session. These three modules will be used to help analyze and improving the overall solution (i.e., the modified minisip and the escrow agent). This sub-project is the topic of **this** thesis.

The second thesis project was conducted in a number of steps:

1. Studying existing IP telephony systems, trusted third party based key escrow encryption systems, and the lawful interception process.
2. Analyzing and finding out the limitations of the above systems.
3. Conducting a feasibility study of the proposed enhancements to create a secure and reliable lawful interception system.
4. Implementing the proposed mechanism of the lawful interception system.
5. Performance analysis in a real-time environment and comparison with other existing key escrow encryption systems with respect to the lawful interception process.

1.5 Outline of the Thesis

In Chapter 2, background study has been presented while Chapter 3 describes some related works of this thesis project. The design analysis of the proposed LI model has been stated in the Chapter 4. Implementation of the proposed model has been described in the Chapter 5. Evaluations and the results have been shown in the Chapter 6. Finally, Chapter 7 and Chapter 8 describe the future work and conclusions respectively.

Chapter 2: Background

This chapter presents some background studies that are required for the reader to understand the whole thesis. As the previous chapter presented the research problem concern a secure IP telephone system with key escrow encryption requires some means to prevent a bad cop in a Law Enforcement Agency from successfully fabricating a recorded session. This chapter describes the basics of an IP telephone System (along with the required protocols for such a systems, specially SIP, RTP, SRTP, MIKEY, and SRTCP). Following this details of LI are presented including (its architecture, existing rules for LI, problems with lawful interception, etc. Finally, the basic of a Key Escrow Encryption Mechanism are presented along with the relevant components of such a system, specifically the trusted third party, public key infrastructure, etc.

2.1 Voice over Internet Protocol (VoIP)

Voice over Internet Protocol (VoIP) is a widely used means of sending voice communication over Internet Protocol (IP) based networks, such as the Internet or other packet switched networks. VoIP is also known as Internet telephony, broadband telephony, voice over broadband, and IP telephony. To make a VoIP call requires setting up a communication session, then in real-time performing the following steps:

1. Sample the analog voice signal and encode it in a digital format
2. Add timing and sequence number information and send this encoded audio via a transport protocol (eventually producing IP packets) to a destination
3. At the destination use the timing and sequence numbers to reorder packets
4. Decode the digitized signal and output as analog audio.

A session control protocol is used to initiate a VoIP call and to decide upon a mutually acceptable audio CODEC. The audio CODEC performs the encoding of the voice signal and decodes the digital data into an analogue voice signal. We will not be concerned with the details of CODECs or impairments to the real-time media stream. Instead we have focused on VoIP with regard to lawful interception (from several points of view – see section 2.3for details).Note that we will focus only on voice calls in this thesis, but the results are also applicable to multimedia sessions.

2.2 Session Initiation protocol (SIP)

The session initiation protocol (SIP) is a widely used application layer standard for initiating, modifying, or terminating a multimedia session between SIP user agents[35][37]. SIP messages are similar to those of HTTP – as the syntax of the message header and status code are reused. SIP uses e-mail style addresses, for example sip:user@kth.se. SIP was designed to be extensible. SIP uses Multipurpose Internet Mail Extensions (MIME) to define the SIP message's contents.

SIP works together with other Internet Engineering Task Force (IETF) protocols such as the Real-Time transport Protocol (RTP), Real-Time transport Control Protocol (RTCP), Session Description protocol (SDP), and others. RTP is used to

transmit real-time data and to provide quality of service feedback (via RTCP); while SDP is used to describe a multimedia session. Details of SIP, RTP, and SDP can be found in [51]. Sections 2.7, 2.8, and 2.9 of this thesis will present some details of secure RTP, secure RTCP, and MIKEY at these protocols are particularly relevant to this thesis.

2.3 Lawful Intercept

2.3.1 General Concept of Lawful Intercept

The term “Lawful Intercept” describes the process by which law enforcement agencies conduct electronic surveillance of circuit and packet-switching communications as authorized by a judicial or administrative order. When the concept of lawful interception was first introduced there was no legal procedure for authorizing interception of communications. Today most countries have adopted legislative and regulatory requirements that providers of public and private communication services (service providers) design and implement their networks with facilities to explicitly support authorized electronic surveillance. International standards organizations have also developed standards to guide service providers and manufacturers about how to implement specific lawful intercept capabilities

Lawful interception is widely used by law enforcement agencies for monitoring network communication. Law enforcement agencies use lawful interception as effective tool for investigating criminal and national security related activities. Lawful intercept is not only used to collect evidence of criminal activity, but can also be used to identify a network of criminals or terrorists. There are two types of information that may be collected: call identifying information and call contents. Generally the requirements to get a lawful intercept order for collecting call identifying information is lower than required for intercepting call contents.

The U.S. Justice Department has defined lawful interception as acquiring call identifying information and/or interception of a call’s contents by law enforcement agencies after having a definite authorization by the relevant governing body or by the court[15][16]. Lawful Interception is a legal and authorized process for secretly intercepting communication by a law enforcement agency or intelligence service [4]. The legal authorization is based on judicial or administrative law. Based upon a lawful order, a network operator, access operator, or service provider carries out the actual interception. This interception of a call’s contents is popularly known as “wiretapping” or “phone tapping” [1][2].

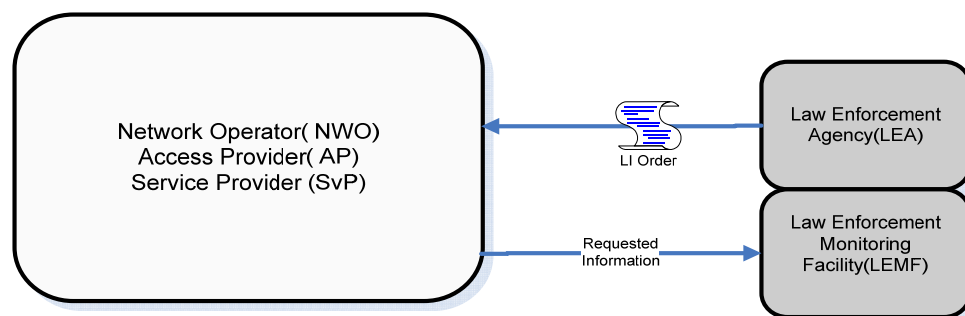


Figure 2-1: Flow chart of a lawful interception (Adapted from ETSI 101 331[2])

A law enforcement agency generally cannot access the communication network directly. Each country has its own laws that apply to telecommunication companies and one or more of these laws set out the specific requirements and procedures that must be followed. For each lawful interception, a law enforcement agency must present a specific lawful interception order to a network operator, access operator, or a service provider to provide access to a lawful interception facility for the purpose specified in the order [5]. This order generally specifies which party (or parties), line (lines), ... are targets of the order and for what period of time the order is effective. Figure 2-1 shows a flow chart of a lawful interception based upon a lawful interception (LI) order and ending with the communication service provider providing the information to the law enforcement monitoring facility.

2.3.2 Reason for Lawful Intercept

Lawful intercept is a useful tool for maintaining law and order within a country. The reasons for lawful interception are varied and vary by country. One of the currently most widely emphasized purposes is for national security, specifically to prevent terrorism. Here there is a great emphasis on learning who communicates with who and when they communicate, in order to learn who is a member of a terrorist network and when there is likely to be some action taken by this human network. Law enforcement agencies also use lawful interception as effective tool for investigating criminal activity, such as cases of cyber stalking, industrial espionage, drug dealing, etc.

2.3.3 Basic Requirements for Lawful intercept

The criteria for lawful intercept differ from country to country. However, there are some common criteria [15]:

- A lawful interception must occur only on a specific target and the subject should **not** be aware of being the target for interception.
- Lawful interception should be transparent.
- During an interception, telephone users must **not** be affected with respect to their service.
- A specified minimum amount of data must be collected and recorded for future use by the law enforcement agency.
- The identity and location of communicating parties in a specific communication should be determined in each interception. In some countries this includes determining their physical location (both for fixed and mobile calls).

An additional criterion for lawful interception in many countries is that no traffic other than the specific target traffic can be captured (although as noted in the next section it may have to be processed to determine that it is not the specific traffic that is targeted). In some countries (such as the U.S.A.) it may be permission to intercept traffic of non-targeted subjects if they are physically near the target – this is generally applied to mobile telephony traffic where a cellular phone is being used by someone else near the targeted subject.

2.3.4 Ways of conducting lawful intercept

There are three ways of conducting lawful interception:

- *With a warrant:* A warrant is issued by the court for a specific lawful interception.
 - *Interception by a Law Enforcement Agency:* After being presented with a warrant the communication service provider provides access to the designated law enforcement agency for carrying out interception. In this case, the interception is conducted by the law enforcement agency.
 - *Interception by the communications service provider:* After being presented with a warrant the communication service provider carries out the interception as specified and stores the intercepted data for the law enforcement agency.
- *Without a warrant:* This is a special type of lawful interception by a law enforcement agency without a warrant. This type of lawful interception takes place for various reasons, such as for terrorism and homeland security.

For various reasons, law enforcement agencies sometimes violate the law with regard to lawful interception. The most commonly stated reason is in conjunction with homeland security. Carrying out an unlawful interception can be punished. (Note that both employees of the law enforcement agency and the communication service provider can be prosecuted for unlawful interception.)

2.3.5 Lawful interception solutions

Lawful interception solutions are classified into three solutions:

<i>Active lawful Intercept</i>	In an active Lawful Intercept, the intercept device directly interacts with the network equipment in order to intercept the specified user's or service's traffic.
<i>Passive lawful intercept</i>	In passive lawful intercept the traffic is sniffed, and then the traffic is analyzed offline to extract the targeted traffic.
<i>Hybrid Lawful Intercept</i>	In a hybrid lawful intercept the network equipment is initially set to passively sniff traffic and analyze it to determine the specific targeted traffic, then the network equipment is configured for active intercepting of the relevant target media streams.

2.3.6 Existing rules and regulations for lawful intercept

Although there are disputes about lawful interception all over the world, most countries have their own defined rules and regulations for lawful interception that allow government agencies to monitor (tele) communication networks. Generally these laws and regulations are designed to enable lawful interception while ensuring citizen's privacy is not violated due to possible abuse of power or misuse by a government employee in a law enforcement agency. (This is the reason why unlawful interception is generally severely punished.)

The rules and regulations for lawful interception vary from country to country due to the respective circumstances of that country. The U.S.A. was the first country that introduced a legal framework for lawful interception. The first law for lawful interception is known as “Title III of the Omnibus Safe Streets and Crime Act of 1968”[51]. Due to concerns about national security, another law was passed to expand the lawful electronic monitoring of communications focused on “foreign intelligence information”[17]. This new act concerns lawful interception within the U.S.A. of “foreign intelligence information”. This act is known as the Foreign Intelligence Surveillance Act (FISA)[17]. Under this act a warrant is not required for lawful interception of radio communication and permits observing both citizens and foreigners. Subsequently a modified version of FISA, called “Communications Assistance for Law Enforcement Act (CALEA)” was introduced in 1994. This legislation compels every telecommunication operator to provide the required facilities and equipment for lawful interception in the U.S.A. It also specified the volume of interception capacity that the operator must provide as a function of the number of lines that they serve (i.e., serving small numbers of lines requires only limited interception capacity, but serving large numbers of lines requires significantly more capacity).

In Europe, the EU directive (95/46/EC)[44] for lawful interception of telecommunication has been depicted as ensuring the privacy of EU citizens. This directive specifies how lawful interception will occur and how the intercepted data will be used. Initially there were disputes among the countries over this directive. Finally, the member states of the EU accepted the directive after an amendment in 2002; the result is known as “Directive 2002/58/EC”. Due to Europeans’ concerns about their privacy, the directive was further modified, resulting in a final version of directive (Directive 2006/24/EC) being accepted by the EU in 2006. This directive specifies the data that can be captured and defines a storage policy for this data. According to this directive communication operators must store data for future investigations by law enforcement. In the case of telephony, this data is usually referred to as a Call Detail Record (CDR). The CDR includes detailed information concerning incoming and outgoing calls. Duration of the data storage depends on each country’s specific policy, but the directive sets a minimum and maximum time period. The storage periods usually range from six months to two years. The directive does not specify who has to pay for this storage, leaving this up to national laws. For example, in the case of Finland the government must pay for the costs of storage, but has free access to the stored data; while in the case of Sweden the operator must pay for the storage and the government must pay to access this stored data.

2.3.7 Problems with lawful intercept

2.3.7.1 Privacy concerns

Individual’s privacy is the main debating point concerning lawful interception. This is because most people consider the communication network as providing a private communication channel for their communication with others. Generally privacy includes the ability of a person or group to select the individuals with whom they wish to share his/her/their information. However, with lawful interception there is no guarantee of privacy in the telecommunication network. Lawful interception is also a threat to the privacy of Internet users – as under the EU Directive Internet

service providers (ISPs) are also subject to both lawful interception and the data storage requirements.

One of the concerns frequently raised in debates on lawful interception is that employees of a law enforcement agency may personally misuse lawful interception, for example for a personal attack by disclosing a target's communication to a third party. Another concern is that the government might misuse lawful interception to target political opponents. Unfortunately there are many examples of both types of misuse.

In the worldwide debate regarding the privacy, some individuals are willing to sacrifice their privacy for greater social benefits – for example, for national security, especially to prevent terrorism. In contrast, other individuals do not want to lose their privacy for any reason. Many countries (including all of the EU, U.S.A., and Australia) have laws concerning the right to privacy. However, these laws do not provide absolute privacy to their citizens, but rather try to achieve some balance between privacy and orderly society. Not surprisingly, privacy is a matter of considerable concern when designing any lawful interception system. Part of the motivation for this thesis project is to re-balance the issue of personal integrity when a lawful interception has been conducted, i.e., to see that information that would be collected to be presented in court would be accurate and that any attempt to tamper with this collected information would be detectable.

2.3.7.2 Vulnerabilities of (and due to) lawful interceptions

Privacy is not the only issue of concern, as the security of the communication system is a significant factor when the network equipment (and software) provides features for lawful interception. Unfortunately, these features for supporting lawful interception may create a security hole -- compromising the network [1]. Lawful interception makes the system less secure in the following ways [1][33]:

1. To intercept the communication in a network, there must be some access points to the network for use by a law enforcement agency. Eavesdropping may easily occur using such an access point.
2. In a trusted third party based system, the trusted third party has access to the master key. Using this master key the trusted third party itself or one of its employees may abuse this trust by improperly disclosing this key or directly utilizing this key themselves.
3. A bad cop in the law enforcement agency may misuse information learned from interception of the communicating parties for his/her own personal interests. (Such misuse includes threatening to disclose information, disclosing information, or using the information gained for personal enrichment.)
4. **Forward secrecy** guarantees that a session key obtained from a durable public and private key pair will not be compromised even if the private key is compromised. In order to ensure forward secrecy, session keys must be destroyed as soon as the communication session terminates. However, key escrow retains the key after the communication session, thus there is an opportunity to break forward secrecy.
5. The trusted third party becomes a prime target for an attacker, since all master keys and/or session keys are stored in a database by the trusted third party.

The main disadvantages of a key escrow system are its design complexity and its ability to scale. For example, if session keys are to be escrowed, then the key escrow system must be capable of supporting both the storage of the aggregated number of session keys and the peak rate of requests to escrow them. Note that the rate of escrowing keys can be reduced by escrowing master keys from which session keys are derived, but this means that a compromise of the trusted third party has an even greater effect and the escrow agent can no longer limit the use of the disclosed master to the derivation of session keys for a limited period of time (as might be required for a lawful interception order). Operating a key escrow system is likely to be expensive and no one has yet introduced a successful business model to support the operation of such a key escrow system.

2.3.8 Lawful Interception Architecture

Although lawful interception is a useful tool for maintaining law and order as well as supporting homeland security, sometimes it is misused. Therefore, lawful interception should be designed in such a way (especially in Europe and U.S.A. where privacy is an important issue to citizens) as to protect against the intentional compromise of communication by an employee of a law enforcement agency. The basic lawful interception architecture is similar in nearly all countries. The first lawful interception architecture standard was introduced by the European Telecommunications Standard Institute (ETSI) and by the U.S.A. The ETSI architecture for lawful interception is depicted in Figure 2-2. This architecture is applicable to fixed line, mobile calls, instant messaging, e-mail, and VoIP calls.

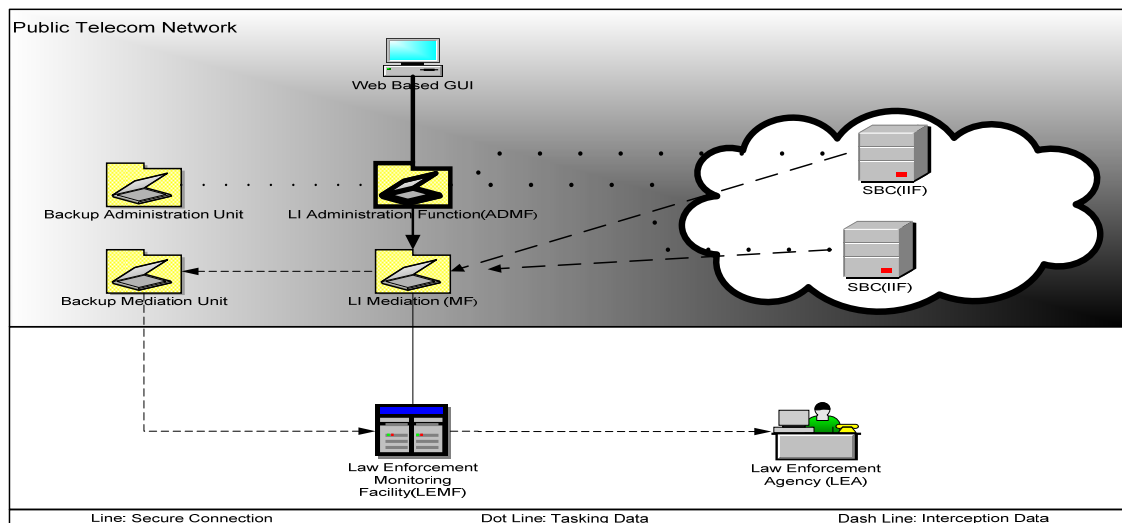


Figure 2-2: HMAC (Adapted from [16][45])

In contrast, after the introduction of CALEA, the Packet Cable Surveillance Model architecture was introduced in the U.S.A. (see Figure 2-3). Call data are grouped into two types: Intercept Related Information (IRI) (used in Europe) or Call Data (CD) (used in USA), and the Contents of Communication (CC). IRI/CD is information about the communicating parties such as the sender's address and receiver's address, duration of the call, and the starting time of the call. In contrast, CC carries the actual contents of the communication.

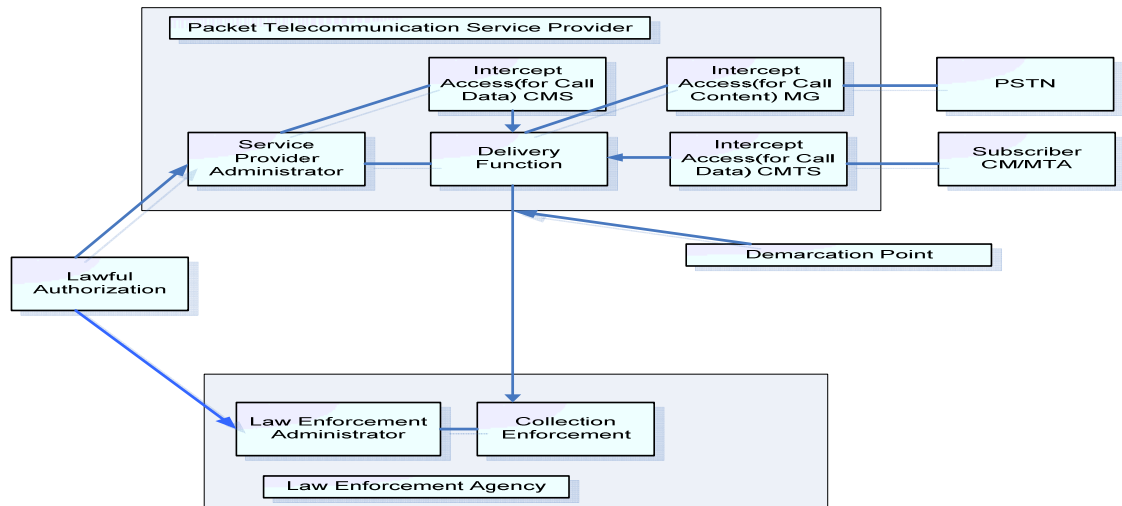


Figure 2-3: Packet Cable Surveillance Model (Adapted from [15][16])

There are three main modules (as shown in Figure 2-3) that are directly involved with lawful interception: an Internal Intercept function (IIF), Administration Function, and a Mediation Function. The Internal Intercept function is placed in the Public Switched Telephony Network (PSTN) operator's node. This is used for collecting target traffic including both IRI and CC based on the lawful intercept order [15][44]. The Administration Function is placed in the Public Telecommunication Network. This is interfaced with the Internal Intercept function and Mediation Function via an Internal Network Interface. The Administration Function controls the interception orders. The Mediation Function is also placed in Public Telecommunication Network, and communicates with Internal Intercept function via an Internal Network Interface and Handover Interfaces (HI2 and HI3) to manage communication with a Law Enforcement Monitoring Facility.

2.4 Trusted Third Party (TTP)

2.4.1 Definition of a Trusted Third Party

A trusted third party is an entity that facilitates interactions between two communicating parties, for example by ensuring the authenticity of credentials. The participants rely on the trusted third party to ensure the security of their communication. For example, a trusted third party acting as a Certificate Authority (CA) might issue a signed digital identity certificate. The CA signs the certificate; this enables others to verify the certificate using the public key of the trusted third party. The trusted third party must not have any common interest with any of the participating parties in order to be trusted by all the parties. Ideally, a trusted third party is highly reliable and recognized by both governments and commercial organizations.

2.4.2 Requirements to be a Trusted Third Party

The G4 security group (consisting of law enforcement agents from Germany, England, France, The Netherlands, and Sweden) has introduced a list of pre-requisites to be a trusted third party. They set forth the following major requirements to be a trusted organization [28]:

- All legitimate users of a trusted third party must benefit from its infrastructure. Integrity, authenticity, and confidentiality must be supported for any financial transactions in an electronic commerce system.
- It should be a universally/widely accepted entity -- supporting both national and international transactions.
- The trusted third party should be a public and unclassified entity.
- Techniques used by the trusted third party should be well known.
- A trusted third party system should support all popular communication techniques.
- The operations of the trusted third party need to be compatible with the laws and regulations of the participating countries with regard to interception, use, supply, and export.
- A trusted third party system must provide access to real-time transactions after presented with a warrant. Additionally, the communicating parties must not know that an interception of their communications is taking place.
- A trusted third party must be assured that a warrant has been issued before allowing a law enforcement agency to access a particular communication session. This access should only allow the law enforcement agency to access the plain text version of the communication, but it should not allow the law enforcement agency access to the subject's master keys.
- The sender must be allowed to limit the time period that a key can be used for a single communication session.
- A trusted third party must support a range of cryptographic algorithms (both in hardware and software).
- Modification of the evidence should be protected by the trusted third party, even though there may be a warrant.
- The trusted third party should ensure non-repudiation, i.e., receiver must recognize the sender.
- A user does not need to deal with a third party in another country. In this case, communication may be established between the third parties, i.e., trusted third parties authenticate themselves and a user will be authenticated to another third party by its own trusted third party.

A trusted third party may be a government agency; a government authorized agency, or a certificate authority (CA). The third party will receive and store the key, while providing it upon being satisfied that the request meets the legal requirements for disclosure. In our implementation we will implement our own trusted third party – simply for testing purposes.

2.4.3 Public Key Infrastructure

A public key infrastructure (PKI) allows communicating parties to share information and securely perform financial transactions through an insecure communication network (such as the Internet) using a public and private key pair. PKI uses a private key and public key pair together with public key cryptography. Public key cryptography is based on the use of asymmetric key cryptography, i.e., the keys are duals – unlike the case of symmetric key cryptography where a single key is used both to encrypt and decrypt messages.

The public key can be made available together with authentication of the ownership of this key signed by a Certificate Authority (CA). This digital certificate can be used to identify an individual or an organization. These digital certificates can be made available to applications by a variety of means, such as a directory service (for example, LDAP). Additionally, the CA can maintain and distribute a certificate revocation list. Thus the validation of a certificate can be explicitly revoked (for example, when a certificate holder or Certificate Authority (CA) is compromised, or a user wishes to prevent the future use of this key). Additionally, these digital certificates have a finite lifetime that is recorded in the certificate, after this time the certificate is no longer valid.

2.4.4 Components of a PKI

A traditional public Key Infrastructure contains of the following components:

- Certificate Authority*** A Certificate Authority (CA) is an entity that is responsible for issuing and verifying a digital certificate. There are different kinds of certificate authorities: User CA, Top CA, and Root CA. The User CA deals directly the user's certificate, while a Top CA deals with the UCA's certification systems. Similarly, a Root CA deals the Top CA's certificate system.
- Public Key certificate*** The public key or information about the public key is added to the certificate. X.509 is a standard for a certificate's structure.
- Registration Authority*** Before issuing a certificate to a user a registration authority performs verification on behalf of the CA. The registration authority is delegated this function with the CA's authorization. A registration authority is a common, but optional, component of a PKI system. The primary purpose of the registration authority is to verify the end entity's identity and to determine whether a certificate is issued or not. This registration authority will impose policies and perform the functions defined in a Certificate Policy and Certificate Practice Statement [30].
- Directories for holding certificates*** The certificate directory is also known as the certificate depository. This is a common, but optional, component in a PKI system [30]. Certificate distribution is performed by publishing each certificate in a certificate directory. The CA or a registration authority manages this directory; in order to simplify certificate distribution. Therefore, the CA simply updates the certificate list, rather than needing to update each entity that might wish to use this certificate.
- Certificate management system*** A certificate management system manages the complete flow of certificate processing. It is used for generating, storing, and verifying certificates. For further details see [31]. Moreover, the structure and functionality of a global certification hierarchy is described in the Internet PEM specification [54].

LDAP is a well-known directory access protocol. It is optimized for providing read access to a database, this is appropriate as there are many requests for certificates in comparison to the number of new certificates. Thin clients can easily use LDAP. An alternative certificate directory access protocol is X.509.

2.4.5 Operation of a PKI

A user (either an individual or organization) first registers with a registration authority and requests a certificate. Usually this registration authority has been delegated by the CA to perform authorization [31]. The registration authority first verifies the requester's information and determines whether the requester has met the requirements to be issued a certificate or not. If the requester qualifies, then the registration authority sends the requester's information to the CA, so that the CA can issue a certificate for the requester. The CA issues the certificate and deposits it in the certificate repository. To issue the Certificate Authority may generate a public and private key pair¹. The private key is communicated to the user through a secure channel and is **not** sent using the Internet or published anywhere in order to ensure its secrecy. In contrast, the public key is added to the certificate and made accessible to all.

When another user wishes to send information to the certificate's user, the information will be encrypted using the user's public key (obtained from the user's certificate as available in the certificate directory). When the certificate's user receives the encrypted information, it can decrypt the information using the private key.

In a similar fashion if a user wants to authenticate information that is to be sent to another user, he or she can sign the information by encrypting the information using his private key. The recipient can decrypt this encrypted information by using the user's public key – since only the user has the corresponding private key only the user could have sent it. The details of digital signatures will be described in the next section.

2.5 Digital Signature

A digital signature ensures a message's integrity. A digital signature is an electronic signature that can be used to authenticate the sender's message. A recipient can use this digital signature to verify that the message was not modified during transmission.

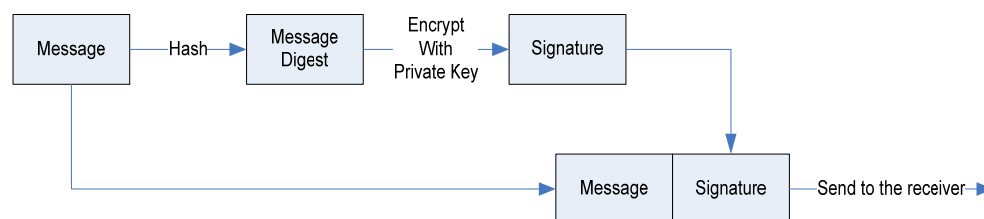


Figure 2-4: Digital Signature Workflow

¹ Note that in some cases the user may generate their own public-private key pair and simply provide the public key to the CA. In this case, the CA does **not** have the private key; hence there is no risk that the CA could disclose it.

Digital signatures are widely used for software distribution, financial transactions, and other instances when forgery or modification of a message might occur. This process is illustrated in Figure 2-4. A digital signature can be used for both plain text messages and enciphered messages. The first step is to compute a message digest by hashing the message using a hash function (see Figure 2-4). This message digest is encrypted using the sender's private key to create the signature. After creating the signature, it is added to the message and sent to the recipient. To verify the signature, the recipient creates a message digest using the same hash function. Using the sender's public key the recipient decrypts the sender's signature to extract the hash of the received message (as generated by the sender) and compares it with his locally generated hash. If the hashes match, then the message is authentic (i.e., it is neither forged nor modified). In this way, message integrity is confirmed by the digital signature.

2.6 Key Escrow

Generally, a cryptographic key (secret key) is not disclosed to parties other than the sender or the receiver. In a *key escrow system* these cryptographic keys are stored by a trusted third party. Escrowing keys is commonly done to protect against the key being lost through accident, death, etc. Key escrow provides an encryption system with a backup decryption capability, thus authorized persons can decrypt the encrypted information using the secret key which is obtained by the authorized party from the trusted third party according to the escrow agreement [16].

Key escrow has also been proposed for use with secure communication systems to ensure that the government can access the encrypted material. The first widely known key escrow system was the Clipper Chip developed by the U.S. Government [36]. Following this, several other proposals have been presented, such as [37][38][39][40]. Some of these may be practical with respect to implementation. However, key escrow systems are considered risky because disclosure of the key may violate the individual's privacy and introduce vulnerabilities into the communication system.

2.6.1 Key escrow encryption system

There are three basic logical components of an escrowed encryption system (see Figure 2-5) [16]: user security, key escrow, and data recovery components. The user security component may be a hardware device or program that performs encryption and decryption along with the key escrow function. A data recovery field may be attached to the encrypted data as part of a general key distribution system. The key escrow component is used to control the storage and release of the data recovery keys. The key escrow component is managed by the key escrow agent and may be part of a public-key certificate management system [16]. The data recovery component is composed of algorithms, protocols, and other tools for decrypting the cipher text. The data recovery component only becomes active when authorized data recovery is necessary [16].

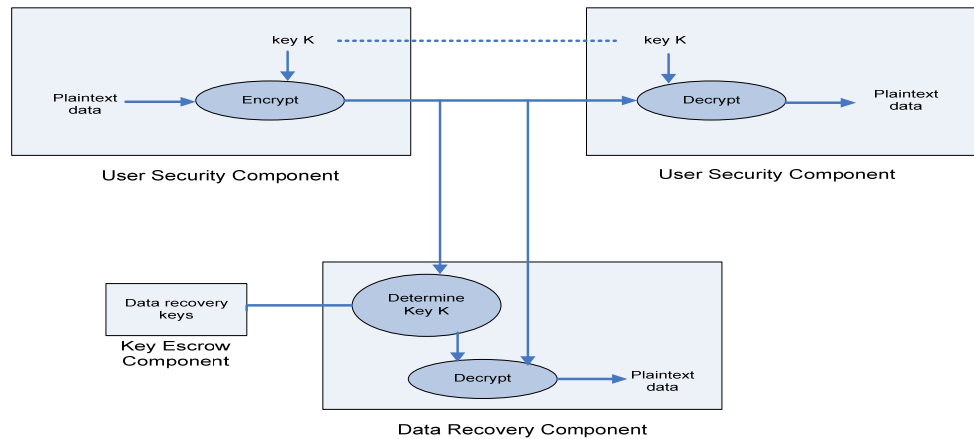


Figure 2-5: Key Escrow Encryption System Components([16])

2.6.2 User Security Component

Following attributes describe the user security component [16]:

<i>Application Domain</i>	A user security component is involved during communication, when processing stored data, or a combination of them. A law enforcement agency uses emergency decryption based upon a warrant for intercepting the target's communications. Stored data might be a simple data file, in this context emergency decryption might be need by the owner to recover lost or damaged keys, or by a law enforcement agent to decrypt the cipher text under a valid court order [16].
<i>Data Encryption Algorithm</i>	Among the characteristics of a data encryption algorithm three characteristics are relevant to the escrowed encryption: algorithm name and mode of operation, key length, and classification of algorithm [16]. The first two characteristics can influence exportability. Moreover, an encryption algorithm can be either classified or unclassified.
<i>Stored Identification and Keys</i>	The user security component stores identifiers and keys for future decryption. Identifiers contain a user or user security component identifier, identifiers for keys, and identifiers for the key escrow component or escrow agents [16].
<i>Data Recovery Field and Mechanism</i>	If a key encrypts data, then the user security component has to bind the cipher text and key with one or more data recovery keys. This binding is performed by adding a data recovery field to the encrypted data.
<i>Interoperability</i>	Interoperability of a user security component is designed for protecting security measures while functioning with a compromised user security component. A user security component is not interoperable with another user security component that does not support key escrow.

<i>Implementation</i>	Hardware, software, firmware, or combination may be used for implementing the user security component. Hardware is preferable for implementing the user security component, since it is more resistant to modification than software. Specific purpose cryptographic processors, random number generators, and/or a high-integrity clock can be part of hardware implementations.
<i>Assurance</i>	Assurance is provided by the user security component for preventing user from creating denial of services of the key escrow mechanism.

2.6.3 Key escrow component

The key escrow component stores all the data recovery keys and assigns the data recovery component to provide the needed data or services. The following attributes characterize key escrow components:

<i>Role in Key Management Infrastructure</i>	The key escrow component can be used as a key management infrastructure. It can operate as an independent key infrastructure or a public key infrastructure. The escrow agent acts as a public key certificate authority.
<i>Escrow Agents</i>	As a trusted third party, escrow agents operate the key escrow component. In order to managing the operation and services for the user security component and data recovery component performed by the escrow agents, they need to be registered with a key escrow center. Escrow agents can be a government entity (used for controlling escrow agent's services by the government) or as a private sector entity (used internally by an organization for commercial purposes). A name and location are required for the escrow agent's identification. Accessibility of the agent is determined by the location of escrow agent and their schedule of operations. Accountability is another characteristic of an escrow agent. The escrow agent also has a certain level of liability. To qualify as an escrow agent, one is required to meet particular standards and to be certified by the government or a globally recognized body.
<i>Data Recovery Services</i>	The key escrow component provides various services, such as release of data recovery keys, derived keys, and decryption keys; and to carry out threshold decryption. The data recovery service is defined by its authorization procedures, services, and transmission of data to and from the data recovery component.
<i>Safeguards for Escrowed Keys</i>	To protect the escrowed keys, a combination of technical, functional, and legal procedures can be introduced. The security of keys is the responsibility of the key escrow component.

Data Recovery Keys

Data recovery keys have the following characteristics:

- Data recovery keys consist of data encryption keys, product keys, user keys, and master keys. A data recovery key might be a session key, network key, or file key. A product key is unique for each user security component, while user's keys indicate a public-private key pair that is used for data encryption. The master keys work only with the key escrow component.
- Storage of keys can be off-line or on-line.
- Keys can be escrowed partly or absolutely.
- Keys are usually be escrowed for specific time period, such as: system initialization period, user registration period, or manufacturing period.
- Key updates may be allowed based upon policy.
- Keys can be generated by the key escrow component, user security component, or combination of these.
- Keys can be split for restoring by the all escrow agents or by a subset of escrow agents.

2.6.4 Data recovery components

The data recovery component can decrypt data from a cipher text message based upon parameters obtained from the key escrow component and the data recovery field. The data recovery component can be defined by its capabilities: data encryption key recovery or data decryption key recovery. Capabilities of a data recovery component are: timely decryption, real-time decryption, transparency, independency, and post-processing.

2.7 Secure Real Time Transport Protocol (SRTP)

The secure real-time transport protocol (SRTP) is an important protocol for protecting voice over IP since it has no effect on the sound quality – while the payload overhead is modest [11]. It is an extension of the real-time transport protocol (RTP) and it provides security features, such as encryption, message authentication and integrity, and replay protection for RTP-based applications. Among these, integrity protection of the message is mandatory (since modification of an RTCP packet might disrupt the RTP streaming process) while other functions are optional and do not depend on each other. SRTP supports both unicast and multi-cast applications [8].

2.7.1 SRTP Architecture

SRTP describes an RTP profile known as Secure Audio Video Profile (SAVP); an expansion of the audio/video profile (AVP) [8][11]. Security features are added to AVP to create SAVP. SRTP operates between the RTP and transport layer. After intercepting an RTP packet, SRTP transmits a SRTP packet to the sending entity. In the reverse direction, after intercepting an SRTP packet, SRTP transmits the

equivalent RTP packet to the receiving party [49]. The encrypted part of an SRTP

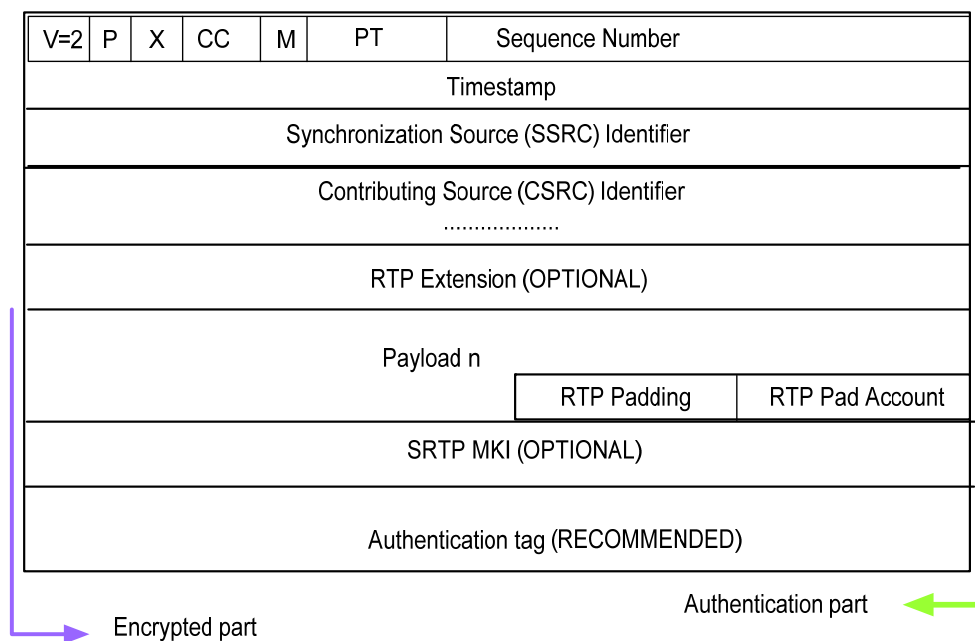


Figure 2-6: SRTP Packet Format (Adapted from[11])

packet contains the encrypted RTP payload of the RTP packet. This encrypted part may be equivalent or larger than the plain text in size [8]. Since predefined transforms do not use any padding, the size of RTP and SRTP packets is similar. If new transforms are included in SRTP, padding may be needed – this would increase the payload’s size. RTP has a specific format for padding that is used as the default method for transform requiring padding. An SRTP packet is shown in Figure 2-6. SRTP consists of all RTP fields and two additional fields: Master Key Identifier (MKI) and authentication tag. MKI is a variable length and optional field. The key management protocol is responsible for defining and controlling this value. The MKI is used to identify the master key that is used to produce the session key. This session key authenticates and/or encrypts a packet. Another field “Authentication tag” is a configurable and recommended field that is required to transform message authentication data. If both encryption and decryption are required, encryption is performed before authentication in the sending participant and vice-versa in the recipient participant. The 16 bit sequence number field in the RTP packet is re-used in SRTP for synchronization. The value of this field rolls over every 2^{16} packets [8][49]. This roll over process requires that the security algorithms re-key. Re-keying may occur frequently for application with high bit rates and short packets [49].

2.7.2 SRTP Cryptographic Context (parameters and functions)

The components such as the peer-packet index, keys, algorithms used to carry out the process, etc. are required by SRTP in order to perform its function. Some of these components function on a per session basis, while others function on a per packet basis. All these combined components are called a cryptographic context. The main components of SRTP are:

<i>Keys</i>	<p>SRTP uses two types of keys such as session keys and master keys. SRTP carries out its functions depending on six session keys which are derived from a master key by a key derivation function. Key Management protocol is responsible for executing the key derivation function that splits the master key into six session keys. A pseudo-random function (PRF) performs the key splitting. The pseudo-random function requires three additional parameters:</p> <ul style="list-style-type: none"> • Key derivation rate as per the configured rate. • The master salt key to prevent key collision attacks. • A label whose value generates the distinguishing session keys. <p>The pseudo-random function is implemented using the Advanced Encryption System (AES) algorithm. Here AES is used in counter mode. Lengths of the six session keys can be customized and are used by SRTP to protect the media. Depending on the security functions, a triplet of keys is required to protect the RTP packet's payload. These keys are: a session encryption key, a session authentication key, and a session salt key.</p>
<i>Rollover Counter (ROC)</i>	The rollover counter is a 32-bit unsigned counter that is responsible for recording the number of times that the 16-bit RTP sequence number has been reset to zero after passing through 65,535.
<i>S_1</i>	S_1 is a 16 bit sequence number which is considered as the highest received RTP sequence number. This parameter is only used by the receiver.
<i>Encryption Algorithm Identifier</i>	This parameter indicates the encryption algorithm and its mode of operation.
<i>Message Authentication Algorithm Identifier</i>	This field indicates the message authentication code.
<i>Replay list</i>	This is only used by the receiver. This component contains the recently received and authenticated RTP packet(s).
<i>MKI indicator (0/1)</i>	This indicates whether an MKI is present in the SRTP and SRTCP packet.

2.7.3 SRTP Algorithms

Selection of a cryptographic algorithm for SRTP is a crucial issue in order to function efficiently. The efficiency of SRTP is measured by the overhead in terms of additional bytes in each message, the run-time efficiency of the code, the code size, and how well SRTP works in a heterogeneous network. Default algorithms are specified for the SRTP in order to simplify the signaling of the algorithms and parameter identifiers, as well as to facilitate interoperability. SRTP provides integrity using HMAC based on SHA1 [8][49]. It is notable that the integrity of RTP means

ensuring the integrity of the RTP header, RTP payload, and the local rollover counter. On the other hand, SRTP uses two stream ciphers to offer confidentiality. Since the SRTP ciphers operate separately on each packet, security processing of one packet is independent from its preceding packet. Hence SRTP handles both packet loss and packet re-ordering. The AES block cipher introduces two ciphers that are distinguished by the mode of AES: counter mode and f8 mode. The input parameters to AES are the session encryption key and an Initialization Vector (IV). The output of the stream cipher is X-ORed bit-by-bit with the plain text to encrypt the data. The default encryption transform is AES in counter mode. In this mode the Initial Vector is composed using the SRTP index of the packet, the SSRC field carried in the RTP packet header, and the SRTP session salt key. An optional cipher in SRTP is AES f8 mode. In this mode, the Initial Vector is composed of an additional RTP header field along with fields used in the IV in counter mode. This added field is used to offer *implicit* authentication.

2.7.4 SRTP Procedure

As described in [8][49], SRTP at the sender's side performs the following steps (assuming that the initialization of the cryptographic context has already been performed through key management):

- Intercepts the RTP packet
- Transforms the intercepted RTP packet into an SRTP packet
- Forwards it to the lower layers for transmission

At the receiver's side SRTP performs the following steps (again assuming that the cryptographic context has already been initialized):

- i) SRTP packet is intercepted
- ii) The packet is processed as per the SRTP rules (which included transforming the SRTP packet into an RTP packet)

Finally the RTP packet is delivered to the application.

2.7.5 Protection provided by SRTP

SRTP provides security both for RTP stream and the associated RTP Control Protocol stream. The protection is offered while RTP packets are transmits over UDP. By encrypting the RTP payload SRTP ensures confidentiality of the RTP packet and by adding an authentication tag to each packet, the integrity of each RTP packet and the packet header are protected. A Master Key Identifier may be added to each packet if multiple master keys are simultaneously in use.

2.8 Secure Real Time Transport Control Protocol (SRTCP)

As noted above, SRTP also provides security for the associated RTP Control Protocol (RTCP) stream in the same way as for the RTP stream. While RTP integrity protection is *optional*, integrity protection is *mandatory* for RTCP.

The design goal of RTP was to deal with real-time media streams that usually use time stamps and buffering. For real time transport two components are needed:

Data Transport and control [13]. RTP provides the data transport and RTCP provides control during a real-time communication session. RTCP information can be used by applications to adapt their network traffic to the available bandwidth. The main function of RTCP is to keep track of received packets and to convey additional information (such as user and domain name, e-mail address, phone number, etc.) regarding the source of the media stream. RTCP also helps to synchronize multiple RTP streams. While RTP packets are generally sent at a high rate, due to the information in the real-time media stream; RTCP packets are normally sent at a much lower rate (typically using a total of less than 5% of the session bandwidth). The RTCP- Extended Report (RTCP-XR) is an extension of RTCP which allows a node to convey detailed voice quality measurements in a VoIP network [11].

The Secure Real-time Transport Control Protocol (SRTCP) builds upon security features of the Secure Real-time Transport Protocol. SRTCP add three additional mandatory fields to the RTCP information, specifically: an SRTCP index, an encrypted flag (E-flag), and the authentication tag; as well as an optional MKI field. The SRTCP Index is a 31-bit field. The E-flag is a one bit required field that identifies whether the current SRTCP packet is encrypted or unencrypted. If E-flag is equal to 1, the packet is encrypted. On the other hand if E-flag is equal to zero, the packet is unencrypted. A 31-bit SRTCP index is mandatory field that performs a counter for SRTCP packet and explicitly added with every packet instead of implicit approach that used in the SRTP packet. Index value is initialized to zero before transmitting the first packet and increased by one for the successive packet. "Authentication tag" is a configurable length required field that requires carrying message authentication data. Master key Indicator (MKI) is also configurable length optional field. It performs similar to the SRTP as depicted in section 2.7.

The encrypted part of each SRTCP packet contains the encrypted RTCP payload of a compound RTCP packet and the range that is encrypted starts from the first RTCP packet to the end of the compound packet [8]. In contrast, the authenticated part of an SRTCP packet includes the whole corresponding RTCP packet, E-flag, and the SRTCP index.

2.9 Multimedia Internet KEYing (MIKEY)

2.9.1 General Concept of MIKEY

To protect a real-time application running over a heterogeneous network using SRTP, a key management scheme is required, as manually managing keys does not scale well. MIKEY is such a key management protocol. It has been introduced for peer-to-peer, simple one to-many, and small size groups to offer a suitable key management system for use with real-time applications [5]. To enable a secure SIP based call between two peers requires that the security is set up by mutual agreement of the parties or that each party sets up the security for its outgoing stream. In the case of a simple one-to-many session the sender is responsible for setting up the security. In a many to many session, a centralize controller unit sets up the security. As a key management protocol, MIKEY features end-to-end security, simplicity, tunneling, and independence from any specific security functionality of the underlying transport protocol. MIKEY is also efficient in terms of its low bandwidth consumption, low computational workload, small code size, and minimal number of round trips. MIKEY produces a Data Security Association for the security protocol which

includes a Traffic-Encrypting Key (TEK) [7]. This TEK is used as input to the security protocol and is obtained from a TEK Generation Key (TGK).

A secured MIKEY session is known as Crypto Session (CS). This session may consist of uni-directional or bi-directional data streams. A pool of one or more crypto sessions is known as Crypto session Bundle (CSB) -- also called a multimedia session. Several media sessions (consisting one or more uni-directional streams, bi-directional video stream, and/or HTTP streams) can be part of a multimedia session. Each Crypto Session has its own unique ID within a Crypto Session Bundle. The concept of a Crypto Session Bundle was introduced for circumstances when groups of one or more crypto sessions have the same TEK Generation Key and security parameters, but the TEKs obtained from MIKEY are separate.

The Group Key Management Architecture (GKMARCH) is a common architecture for a group key management protocol. As a part of this architecture, MIKEY is used as a registration protocol. The essential objects of this architecture are Group Controller/Key Server (GCKS), the sender(s) and the receiver(s). Sometimes a sender acts as a GCKS in MIKEY to forward keys to the receiver. Under specified conditions, MIKEY uses the same transforms as used for SRTP, such as [7]:

- Key data Transport encryption: AES-CM
- Hash functions: SHA-1
- Pseudo random number generator: SHA-1
- MAC and verification Message function: HMAC-SHA1

2.9.2 MIKEY Key Management Procedure

A crypto session bundle is created when a group of security parameters along with the TGKS are agreed upon. Each TGK helps to create a corresponding TEK in a secure way. The TEK and security parameters are combined to create a Data Security Association (SA). This Data SA is input to the crypto security protocol. Depending upon the security protocol, the TEK is used in one of two ways: (i) the TEK can be used directly by the security protocol or (ii) session keys can be generated for TEKs. The transport exchange phase of the MIKEY protocol can be used further to derive new TGKS or specific security parameters. After the transport exchange phase, MIKEY can modify the TEKs and crypto session in a crypto session bundle. During the key transport exchange phase, MIKEY uses the following five key agreement methods to establish a common secret (these methods are described in detail in section 2.10):

- Pre-Shared Key
- Public Key Cryptography
- Diffie-Hellman
- DH-HMAC (HMAC authenticated Diffie- Hellman)

For providing point-to-point security between the conversing parties, MIKEY messages are encrypted and integrity protected. Thus MIKEY generates keys in order to encrypt the message and the security parameters that will be signaled in-line. The key data transport payloads (also known as KEMAC) consist of encrypted key data sub payloads. A KEMAC may have several key data payloads and the last key data

sets its NEXT PAYLOAD field to the value LAST PAYLOAD. In Figure 2-7, a payload in MIKEY is depicted.

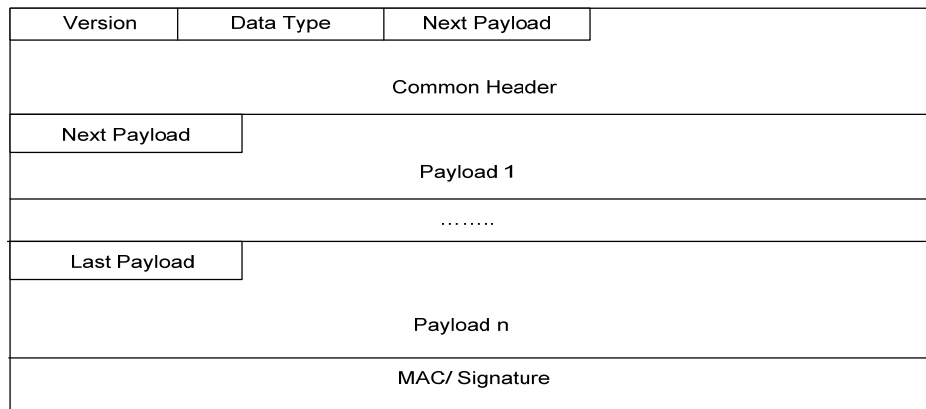


Figure 2-7: Structure of a MIKEY Message (Adapted from [7])

The NEXT PAYLOAD field is an 8 bit field that indicates the subsequent payload. The 8 bit ENCR ALG field indicates the encryption algorithm that was used to encrypt the data field, ENCR LENGTH is the length of the data in bytes, ENCRYPTED DATA is the encrypted data, MAC ALG indicates the authentication algorithm used, and MAC/Signature is the message authentication code for the entire message. Session initialization protocols (such as SIP, SAP, RTSP etc.) can transport MIKEY. 3GPP uses MIKEY in a standalone mode [5].

2.10 Key Agreement Schemes

2.10.1 Pre-Shared Key

In a pre-shared key-agreement protocol, the two communicating parties, client1 and client2, share a secret key through a secure channel before establishing communication. The secret key is denoted as S . From this secret key S , both participants generate an encryption key K_e . As a sender client1, creates a session key $S_{session}$ and encrypts this session key using the encryption key K_e and send it to the receiver (client2). For authentication, an authentication key K_{auth} can also be derived from the shared secret key S . A MAC is generated using this K_{auth} .

2.10.2 Public Key Cryptography

Public key cryptography is a widely used method that underlies several internet standards such as Transport Layer Security (TLS)[23], Pretty Good Privacy (PGP)[24], GNU Privacy Guard (GPG)[22], etc. The distinguishing feature of public key cryptography compared to symmetric key cryptography is the use of an asymmetric key algorithm instead of symmetric key algorithm or in addition to a symmetric key algorithm. In an asymmetric key algorithm, the key used for encryption is not used for decryption. For asymmetric encryption a key pair is generated. The public key is generated based on a private key and some shared constants. If K_{priv} is the private key of a communicating party (or device) and C is a

pre-shared constant known by all communicating participants, then the public key can be generated using function $F(K_{pvt}, C)$.

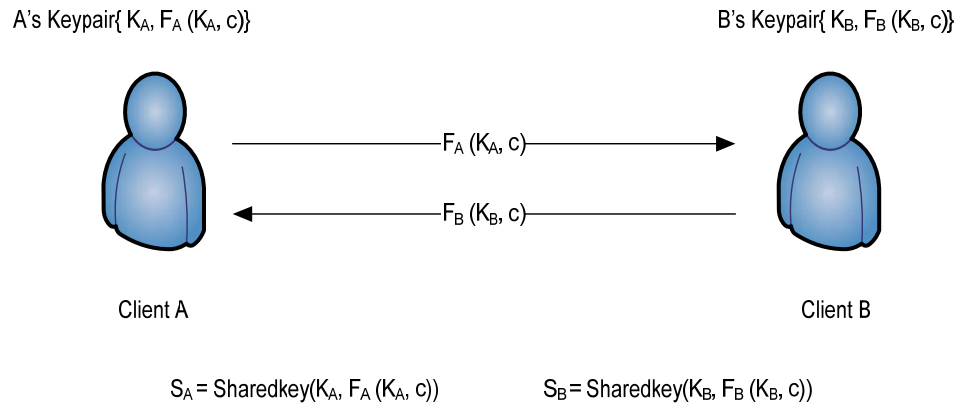


Figure 2-8: Key Exchange using Public key Cryptography

For example, consider communicating peers (clients) A and B. A has the private key K_A and public key $F_A(K_A, C)$. While B has the private key K_B and public key $F_B(K_B, C)$. A calculates the shared key after getting B's public key $F_B(K_B, C)$ using its own private key K_A . Thus the shared key generation algorithm "sharedKey" generates a shared key for A as:

$$S_A = \text{sharedKey}(K_A, F_B(K_B, C))$$

Similarly, B calculates the shared key after getting A's public key $F_A(K_A, C)$ using its own private key K_B . Thus the shared key generation algorithm "sharedKey()" generates the shared key for B as $S_B = \text{sharedKey}(K_B, F_A(K_A, C))$.

The function "sharedKey()" uses an algorithm such that the shared key generated by both A and B will be identical, i.e. $S_A = S_B$.

Two main features of public key cryptography are:

- Confidentiality** A message encrypted by the recipient's public key cannot be decrypted by others *unless* they have the private key.
- Authenticity** If someone has access to the sender's public key, then he or she can verify that the message is signed by the sender's private key. Thus public key cryptography ensures the authenticity of the message and that the message has not been modified.

2.10.3 Diffie-Hellman

The Diffie-Hellman key agreement protocol is used when two communicating parties have no prior knowledge of each other, but they need to establish a shared secret key through an insecure channel [21]. In this protocol two communicating parties (client₁ and client₂) first agree on two shared parameters p and g , where p is a prime number and g is an integer that is less than p . Both of these numbers are assumed to be both public (hence they could be used by any user). Client₁ chooses a random number for his private key and client₂ also chooses a random number b as his private number. Client1 calculates:

$$A = g^a \pmod{p}$$

and sends A to client₂. Similarly client₂ calculates:

$$B = g^b \pmod{p}$$

and sends B. After that both client₁ and client₂ compute their shared key as

$$K = g^{ab} \pmod{p}$$

In this case client₁ computes K as:

$$K = B^a \pmod{p} = (g^b)^a \pmod{p} = g^{ab} \pmod{p}$$

And client₂ computes K as:

$$K = A^b \pmod{p} = (g^a)^b \pmod{p} = g^{ab} \pmod{p}$$

This K is used by client₁ and client₂ to share their information.

2.10.4 DH-HMAC (HMAC authenticated Diffie- Hellman)

HMAC authenticated Diffie- Hellman is a modified version of Diffie-Hellman MIKEY. This protocol uses Hashed Message Authentication Codes (HMACs) instead of certificates and an RSA signature to authenticate the communicating participants to each other [24]. The following parameters are used for the key exchange:

x_i	Secret, (pseudo) random Diffie-Hellman key of the Initiator
x_r	Secret, (pseudo) random Diffie-Hellman key of the Responder
ID_i	Identity of initiator
ID_r	Identity of receiver
DH_i	Public Diffie-Hellman half key g^{x_i} of the Initiator
DH_r	Public Diffie-Hellman half key g^{x_r} of the Responder
SP	MIKEY Security Policy (Parameter) Payload
T	Timestamp
TEK	Traffic Encryption Key
TGK	MIKEY TEK Generation Key, as the common Diffie-Hellman shared secret

Key exchange in DH-HMAC occurs as in Figure 2-9 [25]. The initiator sends a HMACed message along with g^{x_i} and timestamp T to the responder by choosing pseudo random value x_i . Initiator also sends the initiator identity payloads ID_i and responder identity payloads ID_r with the initiator message. Then the communicating parties can calculate TGK as $g^{(x_i * x_r)}$. To avoid man-in-the-middle attack, the HMAC authentication provides authentication of the DH-half keys. Since both communicating parties calculate one exponentiation and one HMAC first and after that perform HMAC verification and another exponentiation, digitally signed Diffie-Hellman is more expensive than DH-HMAC.

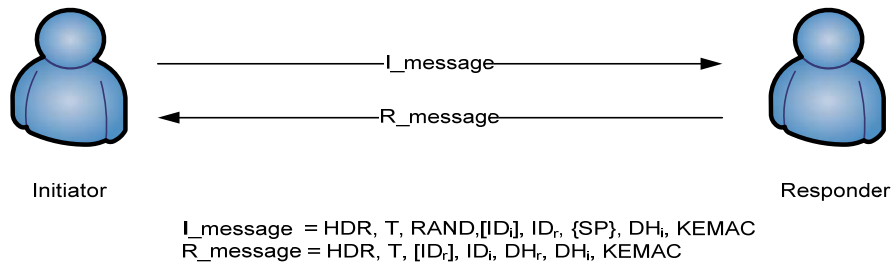


Figure 2-9: Key Exchange using DH-HMAC

2.10.5 RSA-R (Reverse RSA)

In Reverse RSA mode the shared secret key is exchanged without any PKI, and public key encryption performs this task [48]. One full round trip is required to accomplish this. In this mode (as shown in the Figure 11), the initiator sends a signed *I_MESSAGE* asking the Responder to send traffic keying material. Here the *I_MESSAGE* consists of the Initiator's CERT or a link to the CERT.

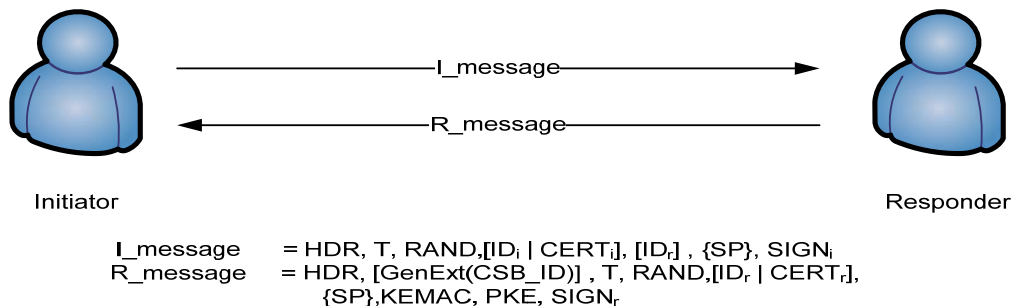


Figure 2-10: Key Exchange in RSA-R

Subsequently a Responder reply with a *R_MESSAGE* that consists of the Responder's CERT or a link to it. To send the encrypted TGK in the *R_MESSAGE*, the Responder uses the Initiator's public key from the CERT learned in the *I_MESSAGE*. While the Initiator uses the CERT in the *R_MESSAGE* to verify the authentication of the actual communicating party and after verifying the Responder the Initiator accepts the TGK. In contrast, for group conferencing using MIKEY-RSA-R, all group members initiate MIKEY with the group key server in order to download secure session information. Either a group key server or a group key sender may act as the responder. In this case, the initiator must **not** send the SP payload. Instead the responder sends all payloads required to introduce a group policy and the payloads that are used for key derivation.

2.11 Minisip

Minisip is a SIP user agent that can be used for making phone calls, instant messaging, and video calls between two (or more) parties in the same SIP network [8]. For security purposes, minisip uses authenticated key exchange by using MIKEY, TLS for signaling, and SRTP for media. Minisip is implemented in C++. It is available as open source code as several libraries under the GNU Lesser General Public License (LGPL) and applications under the GNU General Public License (GPL) agreement [8]. Minisip's features were mainly developed in eight projects, where each project contains several classes for implementing a specific set of technical features. This is a huge project and describing all the feature of minisip is

out of the scope of this thesis. Further details of the overall design of minisip can be found in the licentiate thesis of Erik Eliasson [17]. Only the parts, libMikey, libMcrypto, libmutil, libminisip, and minisip have been used to carry out this thesis project.

2.12 Wireshark

Wireshark [19] is a widely used tool for network traffic analysis. It is widely used by the network administrators, network engineers, developers, and by anyone who wants to know about the details of network traffic in detail. Wireshark is regarded as one of the best open source network analyzers available today. Like other network analyzers, Wireshark can capture network packet in real-time and can decode the captured packet contents in detail. This can be used both by a bad cop and a good cop. Wireshark can also be used to measure network performance as each captured packet is timestamped. Wireshark can also perform several types of statistical analysis of the captured packets. Wireshark runs on both UNIX and Microsoft's Windows. Features of Wireshark include [19]:

- It can capture network packets in real-time from different media.
- It has many protocol decoders.
- It can import files from other network capturing tools; as well as exporting captured packets to other tools.
- Moreover, one can analyze packet contents in a variety of ways.

A bad cop could use Wireshark for secret key recovery, sniffing, packet injection, etc. In our thesis project, we have used Wireshark to capture and analyze the RTCP and SRTP packets. Using this captured traffic we can extract the encrypted payload, keys, and other information. Additionally, a recorded session can be modified by deleting packets; we have used wireshark to do this. In order to the libpcap file captured by the wireshark, a standard library *libpcap* [60] has been used. Five functions from this library have been used in this thesis project to deal with the libpcap or tcpdump file:

<i>pcap_open_offline()</i>	This function opens a tcpdump or libpcap file captured by Wireshark for reading.
<i>pcap_loop()</i>	This function calls a user defined function to process each packet read from the tcpdump or libpcap file until it detects the end of file.
<i>pcap_dump_open()</i>	This function opens a libpcap file for writing.
<i>pcap_dump()</i>	This function writes a packet into a tcpdump or libpcap file. Before calling this function, <i>pcap_dump_open()</i> function must be called.
<i>pcap_dump_close()</i>	This function closes an open libpcap file.

Chapter 3: Related Work

Many researchers have studied the VoIP security, lawful intercept, and key escrow mechanisms. This section describes some of the most relevant research to this thesis project.

3.1 C. Hett, et al.

C.Hett, et al. have presented a proposal to provide security of RTP packets over the transmission line by splitting the VoIP stream into intervals and by building a cryptographic chain [33]. Their security model splits RTP streams at an interval of adjustable length – based on the time, number of packets, or any other criteria. Hash chains are used to protect the RTP packets from an attacker (who can modify or truncate the message interval) during a conversation between two peers. Every interval has a hash of the last interval including its signature. This interleaving of signatures and hashes confirm that there is a constant stream of signatures that builds an indestructible chain. The goal of their proposal is non-repudiation of a conversation between a caller and a callee.

3.2 Rafael Accorsi

Rafael Accorsi, in his survey [47], presents several Log Data protocols for digital evidence. He claims that log data are increasingly used as evidence in judicial disputes, although existing logging protocols are insufficient. In his analysis he shows that each protocol stores log data in a different format for a future audit. He translated Cohen's trustworthiness into security requirements which will ensure the authenticity of the log data. He classifies the requirements of the log data into two phases: transmission of the log message and storage of the transmitted message.

Rafael Accorsi has proposed an approach based upon secure logs [48]. In the first phase, a log message travels through the communication network between a device and the collector and the requirements for this phase are:

- *Origin authentication*: To ensure that a log message was sent by the authorized device.
- *Message confidentiality*: To ensure that a log message is not being disclosed during transmission.
- *Message integrity*: To ensure that a message is not be modified during transmission.
- *Message uniqueness*: To ensure that each log message should be logged only once.
- *Reliable delivery*: To ensure that each log message reaches the collector.

During the storage phase the following requirements are needed:

- *Entry accountability*: To ensure that log entries include information about the device and the collector.
- *Entry integrity*: To ensure that audit trails cannot be modified after the log data is entered.

- *Entry confidentiality*: To ensure that the log data is not entered in clear text.

All logging protocols should follow these requirements during their design and implementation in order to enable them to provide digital evidence. The “syslog-ng” protocol is a trustworthy log data protocol with the backward compatibility [46][47]. It uses TCP as its transport protocol, supports IPv6, and communication is tunnel via TLS. Another syslog protocol named “syslog-sign” uses a cryptographic signature block with the sent message [46]. To do this it first generates a hash of each message block, and then signs the block. The sender then sends both the message and the signature of that block along with the signatures of the previously sent message. The necessary cryptographic keys are generated during an initialization phase.

3.3 V. Stathopoulos, et al.

In [48], Stathopoulos et al. present a logging protocol for public communication networks. The goal of their work is to protect against an insider attack, since assessing the trustworthiness of an employee is difficult. In an insider attack, the impostor could reconstruct part of the audit trail or log data without any possibility of detection. They apply an approach similar to “syslog-sign” with the addition of a “regulatory authority” to grantee that the logging is performed according to the protocol. In this protocol the regulatory authority receives a signed block periodically from the collector for future use. In the future if there is a need for verification of specific data, the stored signed block stored by the regulatory authority can be used for the comparison with the signed block stored by the collector. The authors claim that in their analysis data forgery is not possible as a copy of the signed block is stored by the regulatory authority.

3.4 Clipper Chip

The Clipper chip was a cryptographic device introduced by the National Security Agency (NSA) of the USA [9][10]. This is the first widely known key escrow mechanism which was developed for protecting private communication while facilitating law enforcement interception of targeted conversations. The algorithm used for cryptographic operation in this chip is known as Skipjack and the key exchange algorithm was Diffie-Hellman. Every Clipper chip had a unique secret key and unique identifier embedded into the device. A session key was generated dynamically and was used to encrypt the conversation’s data. The device’s secret key was encrypted using the device's secret key and transmitted before the start of the session. The encryption key was stored with one or more trusted third party for potential later use by a Law Enforcement Agency to decrypt any conversation.

Matt Blaze claimed in his published paper that the Clipper chip was vulnerable to mis-use. This chip transmitted a 128 bit field which was known as the Law Enforcement Access Field (LEAF). This field contained the information necessary to recover the session’s encryption key. Additionally a 16 bit hash was also included with the LEAF to prevent the forgery of the LEAF field. Blaze found that a brute force attack is sufficient to generate a new LEAF field that would have a valid hash, but would not contain the encrypted session key. Hence this LEAF data could not be used to recover the encrypted contents of the session. This was possible because a 16 bit hash was not sufficiently strong to prevent replacement of the real-session key with another value. Subsequently the Clipper Chip was abandoned by the NSA.

Chapter 4: Design Analysis of the Proposed Model

The goal of this thesis project, relevant background studies, and related work has been stated in the previous chapters. This chapter presents a design analysis of a Trusted Third Party based Key Escrow method that will prevent the bad cop which access to a recorded lawful interception from modifying a recorded session without being detected and thus will protect the integrity of the user's conversation. The methods, parameters, and the advantages and disadvantages of four modules: Escrow Agent Module, module for the Law Enforcement Agency (LEA), a Validation module, and an attacker module will be examined. The first three modules are required to establish a TTP based key escrow mechanism that would prevent a bad cop from misusing lawful interception. The fourth module has been used to validate the proposed approach.

4.1 Escrow Agent Module

4.1.1 Required Fields for Escrow Agent Module

The Escrow Agent (EA) maintains a database in which it stores information to be escrowed for some party. This party may subsequently be a target of a LEA intercept for a particular session. Initially we found that six fields needed to be stored in a record corresponding to each session. These fields are: User ID, TGK, RAND value, CSBId, Signed Hash of the last block, and the time of escrowing [7][11]. The first five fields will be escrowed by the User Agent (UA), while the sixth value contains a locally generated timestamp. The User ID is the user's SIP URI or another URI of the UA. The TGK is the TEK generation key and it has a variable length. RAND is a 128-bit or more pseudo-random bit string sent by the initiator in the initial exchange. CSBId is the Crypto Session Bundle ID and it is a 32-bit unsigned integer [7]. Table 1 shows the parameters that should be escrowed.

The Crypto session ID (CSId) is an 8-bit unsigned integer [7]. The CSId value is initialized when a cryptographic session is established between two UAs following the key agreement protocol. In minisip, the value of CSId is initialized after the MIKEY-SRTP mapping [8]. This mapping occurs at the beginning of the dialogue when the key exchange occurs. Values that are used during this mapping are: policy number, ROC, and SSRC. The policy number is fixed, as minisip currently uses only the SRTP protocol [11]. The ROC is calculated from the SSRC [8]. Initially, the ROC is set to zero. When the sequence number exceeds 2^{16} , the sequence number is reset to zero and ROC is incremented by one. Therefore the ROC will be zero for the first 2^{16} packets of the captured file. It will increment ROC by one when a packet sequence number is found to be zero again in the same captured tcpdump file. The ROC of that packet will be one plus the previous packet. For this purpose a packet will be identified by the time together with the sequence number.

Using the SSRC, the appropriate CSId is identified by the sender module and receiver module of both initiator and responder. For the Initiator, CSId value for the

sender module is 1 and receiver module 2. For the Responder, CSID value for the sender module is 2 and receiver module 1.

If the LEA captures the complete session, then the LEA can tell who is the initiator (and/or responder) and CSID value will be easily identified. In contrast, if LEA starts wiretapping after the start of a conversation, how will be CSID value be determined? As a third party, the EA has no information other than the escrowed information regarding the cryptographic session between two UAs, thus it does not know who the initiator was. One of the thesis goals was to reduce the number of parameters to escrow, but to escrow the information that would be required to generate the session keys from the TGK. This analysis leads to the minimal set of parameters that must be escrowed-these are stated in the Table 1.

Table 1: Parameters that should be escrowed

Escrow Parameters	Currently Escrowing	Escrow in Future	Reasons
CSBID	Yes	Yes	This is generated by the initiator and transformed with the responder during initial conversation.
RAND	Yes	Yes	This is generated by the initiator and transformed with the responder during initial conversation.
CSID	No	Yes	To identify whether the target is initiator or responder.
Signed Hash	Yes	Yes	To determine whether any packets are inserted after the end of the session.
Policy Number	No	Yes	If more protocols than SRTP are used.
Seq. No. of 1 st SRTP packet	No	Yes	It will detect forgery that inserts packets before the session actually started.
User ID	Yes	Yes	To identify the target user session.

4.1.2 Escrow Agent Database

The escrow agent utilizes a database for storing the escrowed information for each session. The EA indexes the escrow information based upon the User Id and the time when the information is escrowed. The EA uses four tables in its database. These

tables are: User Authentication table, Escrowed Information Table, LEA Authentication table, and LEA request table. Each of these tables is described below.

Table Type	Table Information
<i>User Authentication Table</i>	The User Authentication table contains a User Id and information about the corresponding user, at a minimum this might contain a password, but might include other information. In our implementation we have used the user's SIP URI as the User Id. When new users are added to a VoIP service, the EA's User Information Table could be updated directly using information provided by the VoIP service operator or the EA might receive this information when a user subscribes to this EA's service. When a UA wants to escrow information, the EA will first authenticate the UA based upon the provided User Id, then the EA will store the information provided by the UA in the Escrowed Information Table.
<i>Escrowed Information Table</i>	The Escrowed Information Table has (at least) the following fields: Escrow Id, User Id, TGK, CSBId, RAND, CSId, Signed hash, and a local time stamp. This escrowed session information will be identified by the User Id and time (i.e., these two fields will be used as an index for this table). In our implementation we have used the SIP URI as the User Id and we have used a 64 bit time stamp using the same format as used for the network time protocol (NTP). We have assumed that the EA is synchronized with an NTP time server; hence all time stamps will have a common global meaning.
<i>LEA Authentication Table</i>	The LEA Authentication table contains the information necessary for authenticating the LEA. Fields in this table will vary depending upon the agreement between the EA and LEA. In our implementation we have used three fields: LEA Id, Agency Address, and password. The Agency Address will contain the IP address of the LEA entity authorized to receive escrowed information, the text name, and street (or postal) address of the LEA. For our implementation we have assumed that the LEA Id is a positive integer, while the other two fields are variable length text strings.
<i>LEA Request Table</i>	When a LEA asks for the escrowed information of a target, the LEA must provide its identity, the court order, and some other optional information. The optional information depends on the laws of the country where the Escrow Agent is operating. The information identifying the LEA must correspond to that stored in the LEA authentication table of this EA's database. The court order and other relevant optional information for a request for escrowed information of a target will be stored for future use. For the EA some information that must be stored is the time when the request was received by the EA and the time when the response was provided. The EA will also need to store information about

which records from the Escrowed Information Table were delivered at the time of the response.

4.1.3 User Agent Identification

As was noted above, the User ID of a conversation peer may be the SIP URI of the User Agent, rather than the user's personal SIP URI. There are several reasons for selecting a SIP URI as the User Id:

- All VoIP users must have a SIP URI and need to register with one or more SIP registrars in order to receive calls *directed to themselves*. If the proposed key escrowing mechanism is implemented, then this SIP URI could be used as the identification for escrowing information with the Escrow Agent.
- The communications cost of adding the SIP URI to the information that the UA escrows is small.
- Since each UA is registered with a SIP registrar, this same authentication mechanism could also be used for authentication when accessing the Escrow Agent. In a SIP registration, the SIP proxy server maintains an authentication database for the registered User Agents. When a User Agent wants to start a session, then the SIP proxy server authenticate it using the SIP URI stored in the SIP URI database². The Escrow Agent could obtain an up-to-date list of the SIP URI(s) of the registered User Agent(s) from the VoIP service provider. To update the User List in the Escrow Agent, the VoIP service provider can send batch updates; i.e. the current Registered User list could be update at a certain time of the day. In this case, after registration, service for a new user will be activated after the next update of the User List in the Escrow Agent.
- Note that a SIP device also has a SIP URI representing the UA itself. In the case of “pay phone” or other device, this might be the only SIP URI that is available. Thus it may not be possible to identify a call with a specific user – only with a specific UA.

4.1.4 Different URIs for User identification with the EA

If the EA uses a URI other than the SIP URI, there are some issues that need to be addressed:

- The EA and the UA have to use a consistent URI to identify the user. If this URI is different from that used by the UA for sessions, then this other URI is additional information that the UA needs to maintain and the EA needs to learn the URI used by the UA during a dialogue between SIP peers; as the

²The issue of authenticating the user before making or receiving a call is outside the scope of this thesis project; but for convenience we assume that users making calls have registered with their SIP service provide (for example, in order to receive incoming calls).

URI that is used for a session is likely to be the identifier that would be used by a LEA.

- If the UA uses its own SIP URI to authentication itself to the EA, then the EA can validate this URI with the VoIP domain (as all SIP URIs in this domain are assigned by the VoIP domain authority). For example, this means that the SIP URI can easily have a PKI CERT that is signed by the VoIP domain authority.

Instead of the user's SIP URI we could use the Call Id to index the escrow information in the Escrow Agent, since the Call-Id is a globally unique identification of a call. This Call Id is generated by a combination of a random string and the soft/hardware phone's host name or IP address. Three fields (To Tag, From Tag, and Call-Id) uniquely identify a peer-to-peer SIP relationship between two parties.

It is not clear if there should be any interaction between the SIP user's VoIP domain authority and the escrow agent. For example, a user might want to choose an escrow agent that has no business relationship with the user's SIP provider - helping ensure that there is less risk of collusion between the two (as that might reduce the user's privacy).

4.1.5 Required parameters to escrow in future

In the future there will be some additional values that need to be escrowed, such as the policy number. In the current implementation the only policy number is 0, as minisip only uses the SRTP protocol. However, in the future several different protocols might be used; hence the policy number will also be needed to be escrowed. Because protocol initialization happens during initialization phase of the conversation between the peers the value will be known at the end of the session, but might not be known before the session and can vary from session to session and from UA to UA (as not all UAs will implement all security policies).

4.1.6 Implementation Principles

The Escrow Agent has two interfaces – one with the User Agent module and another with the LEA module. After arrival of the escrowed information from the User Agent, the Escrow Agent module verifies the authenticity of the received data and stores it in the database table. In our implementation, the SIP user ID and corresponding password have been used for authentication.

Similarly, when requesting escrow information of a target, the LEA must submit some mandatory information to the EA, such as the agency identification of the LEA, a surveillance order, target information, surveillance type, and court warrant[57]. Any user of the LEA with valid authentication can access the EA module using a web browser. After authenticating the LEA and court order, the EA module will deliver the escrowed information concerning the target during the time period indicated in the court order³. This may result in the escrow agent releasing records of escrowed sessions either in a batch or in a stream, depending upon whether the ending time is before or after the current time. In the event of a stream of records, the court order

³How the EA authenticates this court order is outside the scope of this thesis project.

may specify a bounded delivery time (for example, requiring that the records be provided in “real-time” or within “1 hour”). For our current implementation we have only implement batch delivery of records, the implementation of a stream of escrowed information has been left for future work.

The information that is to be provided by the user or by a LEA will be entered into a web form and the response will be a web page. Note that this access in many case will be via a programmatic interface and not via a web browser.All communication from users, LEA, and UAs to the web service will utilize TLS to protect the communication. Mutual authentication of all the parties in a TLS session will be based upon certificates. For testing purposes we have used self-signed certificates that are manually issued and directly provided to the various participants. The provisioning of certificates is assumed to occur before there is any communication, hence the details of this process lie outside the scope of this thesis – but should be addressed in a future thesis. The implementation has not implemented certificate revocation lists or any other form of checking whether a certificate is still valid – only that the certificate was valid for the time it is being used.

4.2 LEA Module

A LEA module was implemented (1) to provide the information required by the EA to release the desired escrowed information for one or more sessions, (2) using this escrowed information the module can decode a previously captured session, and (3) the module can check if the captured information has been modified (based upon the signed hashes generated by the senders during the session).

4.2.1 Required parameters for the LEA module in order to provided the information required by the EA

The LEA module generates the session keys from the TGK along with other Security Association parameters. For this reason the LEA requires:

- TGK, RAND, CSBId, and CSId from the escrowed information provided by the EA.
- SSRC and Sequence number obtained from the packet header.
- ROC from the SSRC.
- The policy number is assumed to be fixed since SRTP is the only protocol currently used.

Moreover, as noted earlier the sequence number of the first SRTP packet must also be escrowed. Because if this sequence number is not sent, it would not be possible to detect in which block a modification had been made.

To get the escrowed information, the LEA will send a request to the EA through a web based application. The EA will authenticate the LEA's identity and verify that the court order is valid, and then the EA will return the information escrowed by the target during the specific time. Given the escrowed information, the LEA module will save this information in a local text file. Subsequently the LEA module will read the captured packets and using the escrowed information provided by the EA the LEA module will generate the relevant session keys from the TGK using the parameters mentioned above. These session keys will be used to decrypt the captured packet payloads.

4.2.2 Possible Trade-offs of the LEA Module

The LEA module is an important module in the Lawful Interception architecture. Its efficiency and reliability are - a function of the time required to analyze the captured data, the security during the LEA operations, the network overhead, and the desired transparency of the LEA module. In our analysis of the proposed LEA module, we have found the following trade-offs:

4.2.2.1 The time required to decode the recorded SRTP packets

Assuming that the SRTP packets have already been captured, then the time delay before they can be decode is a function of the time to get the court order, the time to get the escrowed information, the time to generate the session keys, and the time to actually decode the SRTP packets.

The time for the LEA to get a valid court order to intercept the contents of the target is out of the scope of the LEA module (and also outside the scope of this thesis). Thus we will begin our timing from the time that the LEA has received the court order. The LEA module needs to authenticate itself to the EA, and then it will submit the court order and other information. The EA will return the escrowed information. After receiving a request, the amount of time taken by the EA to generate a response – depends on the EA module's efficiency and the time required for the LEA module to transmit the information required for the request and the time for the EA to transmit the reply, along with the delay due to the communication channel. If we assume that the LEA module and EA utilize a TLS protected connection, then we have to add to the above delay the processing time for mutual authentication plus the time to transmit the authentication information and challenges and responses between these entities.

$$t_{res} = t_{TLS} + t_{EA} + \sum t_{transmission_delay} + N_{round_trips} * RTT$$

It is not possible to reduce the required time for establishing the TLS connection (t_{TLS}) due to the number of required round trips, as the number N_{round_trips} required establishing a TLS connection is fixed. Therefore, a question arises as to whether we should keep the connection open between the EA and LEA module or not. In this case, two options can be considered:

- Since communication between EA and LEA will not be frequent, if the LEA works offline, it is not necessary to keep the TLS connection open. Instead, a new TLS connection can be established for each instance of escrow information retrieval.
- On the other hand, if the communication between EA and LEA is often, the secure pause and resume session by TLS [58] can be utilized to re-establish the TLS connection once it has been established. After providing the escrow information, the open connection between EA and LEA module can be paused. When a new connection is required between these two modules, both modules can search for the paused connection in their cache and after finding it they can resume the connection in a secure

manner. The design and implementation of this procedure is out of the scope of this thesis, but should be considered in future work.

The time required for responding by the EA (t_{EA}) will depend on how the system is implemented. For example, if the EA module stores the escrowed information in a distributed system, t_{EA} must include the time to reconstruct the information from its distributed elements (Note that the details of the EA's implementation of its database are outside the scope of this thesis.).

Transmission delay ($t_{transmission_delay}$) includes the preprocessing time required to transfer packets to/from the wire [59]. In a packet switching based network, a store-and-forward mechanism is used to store the packets in the buffer and then checks for errors before forwarding the packet. In addition to the processing time, we must also include the propagation delay for sending the packets across the link. Note that this transmission delay includes the time to transmit all of the response to the LEA. As noted earlier we are only considering batch requests & responses, hence the amount of data that must be transferred may be substantial.

Next we have to consider the time required by the LEA module to calculate the session keys from the TGK and add to this the required time for decoding the captured SRTP packets with these derived session keys. We can divide these two steps into: Time required to derive session keys (t_{kd}) and the time required to decode SRTP packets using the session keys ($t_{decode} = t_{per_packet_decode} * N_{packets}$).

The sum of all these delays is the total time required to decode the captured SRTP packets:

$$t = t_{TLS} + t_{EA} + \sum t_{transmission_delay} + N * RTT + t_{kd} + t_{decode}$$

Therefore, efficiency of the LEA module is inversely proportional to this total delay. If the Law Enforcement Agency is doing this processing offline, there may not be a big concern for the total time required. This time can also be speedup by doing parallel processing – as the decoding is highly parallelizable since (1) the packets are encrypted independently they can be decrypted independently and the resulting audio (and video) combined together and (2) separate sessions can be decoded in parallel. Given that the UA that was actually receiving the SRTP packets needs to decode the packets in real-time, it seems reasonable to expect that the LEA should be able to decode captured traffic at least as fast as real-time and perhaps many times real-time speed. It also seems reasonable to expect that as the UA that actually participated in captured session also had to derive the session keys that the time required by the LEA will be comparable (assuming that the LEA can use a processor that is at least as fast as the processor used by the UAs). As the information being sent to and from the LEA to the EA is relatively a small amount of data per session, the time required to transfer this data should be small, but might not be if there are many sessions during the period of time that is subject to the request.

4.2.2.2 Security of the LEA module

The LEA module needs to protect the previously escrowed information that it received from the EA module. It must also protect the derived session keys and the decoded contents of the captured SRTP file. While it might seem that the captured

SRTP (and SRTCP) traffic does not need to be protected especially well (because it is encrypted), unfortunately this is not the case since the time stamps, source and destination IP address and port addresses are **not** encrypted – this information could be used to learn when and where the intercept was taking place, whose traffic was being intercepted, etc. Additionally, if the results are to be presented as evidence in a legal proceedings there may be a need to use secure logging techniques (as described in section 3.2) to preserve the “chain of evidence”. A complete security analysis of the security of the LEA module is outside the scope of this thesis, but should be considered in future work.

4.2.2.3 Network overhead

If the LEA module works offline and the LEA makes only infrequent requests for escrowed information from the EA, then the network overhead of this communication is not expected to be a significant problem. However, if there is a need for real-time (or near real-time) interception, then there are two problems (1) the key escrow architecture that has been used in the minisip client – prevents real-time interception – since the key information is only escorted after the session has been terminated and (2) there is no requirement that the UA escrow this information immediately after the call terminates. Hence network overhead in the communication between the LEA module and the EA module should not affect any solution. Note that this situation would change if there was a requirement for the UA to escrow the material as soon as it was available to the UA; however, further consideration of this lies outside the scope of this thesis project.

4.2.2.4 Transparency of the LEA module

Transparency of the LEA module depends on its design and implementation. The LEA module should be designed and implemented so as to avoid (or discourage) misuse of the system and to avoid forgery or modification of the captured and decoded data. (See the note above concerning providing an evidentiary chain.).

4.2.3 Implementation principles of the LEA module

A LEA module consists of three sub-modules:

1. A capturing module that will capture a target session.
2. Escrow information retrieval module that will communicate with the EA and retrieve the relevant escrowed information.
3. A session decryption module that will derive session keys from the TGK and other information; then decrypt the recorded session with this session’s keys.

The web based application used the LEA module to communicate with the EA can be developed using any web server language such as ASP, CFML, PHP, JSP, etc. The LEA module that will generate the session keys from the escrowed information and that will be used to decrypt the packets will be implemented in C++. We assume that wireshark was used earlier to capture a session; hence the “libpcap” library will be used to read the file containing the captured packets.

There are two major alternatives for implementing the decryption of the packets. Either this could be implemented as a plug-in for wireshark or it could be implemented as a stand-alone program. Additionally, the code could either be written

from scratch or the minisip code could be re-used (since this is essentially the SRTP receive part of minisip's UA code).

4.3 Attacker Module

The attacker module cannot be designed in a fixed way, as there are many potential forms of attacks. In our first implementation, the attacker module will receive voice from an external device, packetize it, and encrypt it as SRTP traffic (i.e., this code will be used to generate a completely fabricated SRTP stream). This fabrication is possible because the LEA module has all of the information necessary to both decrypt and to encrypt the SRTP traffic.

A “bad cop” could replace the original packet payloads with this newly encoded set of packet payloads. The LEA module will of course be able to decrypt this traffic and will output the same audio that the “bad cop” used to generate this traffic. However, the attacker does not have the information necessary to generate correct signed hashes, thus it should be possible to detect such a case of data fabrication.

The attacker module will be implemented in C++. This module will also use the “libpcap” library. As with the LEA module it may be possible to use the existing minisip libraries for most of this (since it is essentially the SRTP transmitter part of minisip's UA code).

4.4 Validation Module

A major focus of the thesis project has been evaluating the performance of the validation module. This module will be used to detect forgery of a captured session. This module will decrypt the signed hashes (captured during wiretapping by the LEA) using the UA's public key found in the UA's PKI certificate. The hash of the decrypted data will be compared to the decrypted hash. The hash generated from forged packets will not match the original hash captured during the session. Note that this module also needs to decrypt captured SRTCP traffic.

4.5 Communication between UA and EA

The UA will escrow the session information with the EA after a TLS [23] handshake between them. This is done to ensure mutual authentication, to avoid a man-in-the-middle attack, and to protect the information that they will communicate with each other. To perform the TLS handshake, the UA and EA perform the following steps **before** any information can be communicated regarding the escrowing of session information:

- First hello messages are exchanged between the UA and EA to agree on algorithms, exchange random values, and check for session resumption.
- The necessary cryptographic parameters are exchanged to allow them to agree on a pre-master secret.
- Certificates and cryptographic information are exchanged to allow the UA and EA to mutually authenticate each other.
- A master secret is generated from the pre-master secret and the exchanged random values.
- TLS can now provide security parameters to the record layer.
- Finally the UA and EA can verify that their communication peer has established the same security parameters.

This result of the above steps is to establish a secure handshake *without* interception by an attacker.

The efficiency of the communication between the UA and EA is important as they work online. We need to consider the amount of time required to perform the above steps and to escrow the required information and the security of the data to be escrowed. We must also consider the scalability of this – particularly from the point of view of the EA.

Since we assume that EA and UA will communicate through TLS handshake, we can separate the total required time into a key exchange time and time for exchanging application data. During key exchange TLS may need several round trips. If the required number of round trips can be reduced, then the communication time (and network delay) would be significantly reduced.

Additionally, frequent calls by (or to) a UA will cause frequent escrowing. In this case, the UA and EA can cache the key exchange information locally once they are initialized. Thus for subsequent calls the TLS session can be quickly re-established, reducing the network overhead, the communication time, and the computational load on both parties.

4.6 Should the EA generate session keys for the LEA

Generation of session keys in the EA module is **not** desirable for the following reasons:

1. The EA is a Trusted Third Party (TTP) and should simply & reliably store & retrieve the escrowed information. Having the EA perform these other computations increases the complexity of the EA, potentially leading to greater security risks.
2. The EA must communicate with all of the UAs that want to escrow information and it needs to do this quickly and efficiently for both performance reasons and for scalability, hence adding additional computational burdens on the EA will decrease scalability.

If session keys were generated in the EA module, there is an increased risk of disclosing the session keys, since more people (both at the TTP and LEA) would know the session keys.

Chapter 5: Implementation of the proposed LI model

This chapter describes the implementation part of this thesis project. Our previous analysis in section Chapter 4: showed that we have to design four modules to achieve our goals. The main system consists of four key components: User Agent, Escrow Agent, LEA module, and Verification module. We have worked with the last three modules. We have also designed and implemented another module (an attacker module to forge recorded session) for verifying and evaluating the performance of the proposed key escrow based Lawful Interception System.

5.1 User Agent

In this project, we have used an open source SIP user agent called minisip, which is being developed at KTH. To implement this module, the minisip User Agent was extended as described in the companion thesis project by Md. Sakhawat Hossen [43]. In this earlier thesis project, a module was added to minisip to create a block of packets of the media stream, compute a hash over this block, then signs this hash using the user's public key and sends these signed hashes as packets. In addition, this module escrows information with the Escrow Agent so that the session key could be re-generated. The module was implemented in C++. For hashing the block, the popular hash algorithm HMAC-SHA1 has been used. For signing the hash the RSA algorithm has been used. The User Agent escrows the key and the other information about the session after termination of a successful session. The details of the modified User Agent can be found in [43].

5.2 Escrow Agent (EA) module

In the proposed lawful interception mechanism, the Escrow Agent is a key components since it works as a Trusted Third Party for storing the TEK generation key (TGK) along with other escrowed information to be used to derive the session keys later. A complete analysis of the EA module's design and requirements has been performed in section 4.1. This earlier section considered the design, parameters, communication between UA module and LEA module, and trade-offs that may be arise during its practical operation in the real world. Based upon this analysis, a simple EA module has been implemented. However, for a more sophisticated EA module, further technical issues need to be addressed; these are considered to be out of the scope of this thesis project.

The EA module is web based and co-located with an Apache web server. It maintains a database named *db_escrowAgent* consisting of four tables (see Table 2). MySQL is used for the database and Apache has been used as the web server. PHP has been used for accessing the web server programmatically. Using this module, an LEA employee login to access the Escrow Agent module (as shown in the Figure 5-1) in this implementation a LEA user name and password are used for this login.

Table 2: Used Database Tables

Table Name	Table Type	Reasons
<i>t_userAgent</i>	User Authentication Table	To store SIP User Agent authentication information
<i>t_escrowed_info</i>	Escrowed Information Table	To store escrow information.
<i>t_lealogin</i>	LEA Authentication Table	To store LEA authentication information
<i>t_escrwoinfo_request</i>	LEA Request Table	To store information about the requested escrowed information

**Figure 5-1: LEA Module Login Interface**

The EA module verifies the LEA employee's login information and returns another web form for the employee to provide the relevant information to get the escrow information along with the TGK. The information required in our simple EA is simply the target's SIP URI and a date and duration in hours, as shown in Figure 5-2. A better version of this interface might have two fields for the time period, i.e., *start of time period: xxxx-xx-xx yy:yy:yy* and *end of time period: xxxx-xx-xx yy:yy:yy*.

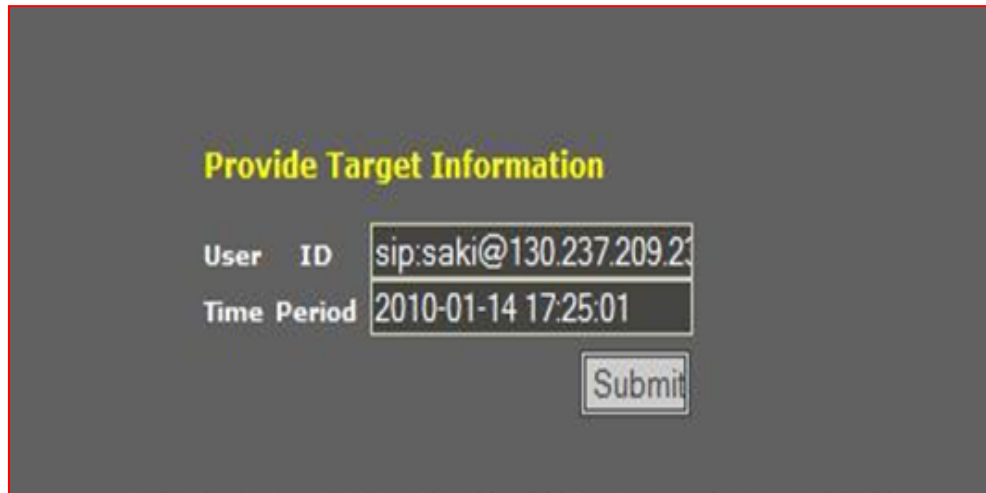


Figure 5-2: Interface for Providing Target Information

After providing the required information the EA module replies with the TGK and other escrowed information (as shown in the Figure 5-3). An actual lawful intercept application would require the LEA user to provide more information, such as Agency identification of the LEA, surveillance order, surveillance type, and court warrant [57]. The exact requirements depend on the specific law for the electronic surveillance in the country where the intercept is to be conducted.

Table 1: User Info

USER ID	RAND VALUE	CSB ID	TGK	SIGNED HASH	TIME
sip:saki@130.237.209.238	QjLmyL5EPrO7F1EiU3sriQ==	-1626928177	S1PhTWf12jFpvejCLCm I5L95Z1Aa3BiWcaqGIDJ 2zldmIPyNqj6T17qzpg 07dI9fZV6dK7GZFFwUL 1sdG9PPpalvL7hU0hM6q hi4IV93/qwOrn6D 2g3muQjPjS69joGJcSDN 85KzV5gTsPnU9kO7PFFA KiN8zvpQa6cUBN787Mnl MZevm6e9YgVqNdrZ4A1Z KA/b4rGT40Nlnq9f41R 9rP81P926GKN0dWUeJ54 Pv3v0IUT9IUFFSrtqoy	oSN/4DI2Q/SXI0G2/QAt Nx qgwdRtswtZMGdz44qm6 yYSQDOuGPuv4gK33gStk qSDmny6HhN62mxyb6ZNL rr5FT4tLFV1BwMAu80Om BPysvfAik3OewhUM6R5l xYRbiWHM50EdC7dbegHZ UT37oMfghKLmudRvYop IAPJRvJM=	2010-01-14 17:25:01

save cancel

Figure 5-3: Escrow Information returned by the EA

In our implementation the escrowed information will be saved in a simple text file on the local machine from which the EA web page was accessed (i.e., the LEA employee’s local machine). The escrowed information should be sent using HTTPS. For testing with HTTPS, a self-signed certificate has been used.

5.3 LEA module

In section 4.2, the design principles of the LEA module were described. Based upon these design principles, a LEA module has been implemented using the algorithms described in the following sub section.

5.3.1 Algorithms of LEA module

The LEA module was implemented using C++. To develop this module, some existing minisip and Open SSL modules are reused. The LEA module performs the following steps:

- The LEA must capture the conversation between two parities. Developing a module to capture an online conversation is out of the scope of this thesis; however, numerous tools exist for this.
- The captured packets must be filtered to include only the SRTP packet and RTCP packet of the intercept target. Filters for SRTP and RTCP packet (as used in Wireshark) are respectively:

```
Ip.src==<ip address>(eg.130.237.15.252)&&rtp && !icmp
```

and

```
Ip.src==<ip address>(eg.130.237.15.252)&&!rtp && !icmp
```

- The LEA module begins by reading the packets from the captured tcpdump file (for example, a file with the extension .libpcap) and re-packetizes them into SRTP format.
- Next the LEA module reads the previously escrowed information from the previously saved text file provided by the EA module.
- The LEA module scans for the Sequence number and SSRC of each packet of the captured SRTP packets.
- The LEA module calculates ROC using the Sequence Number and SSRC value. For ROC calculation, we assume that LEA captured the full conversation between two communicating peers for any electronic surveillance. An algorithm to perform this computation is depicted in section 5.3.2.
- A cryptographic context is computed based upon the previously escrowed information, ROC, Policy number, and CSId. This cryptographic context is generated according to the MIKEY-SRTP protocol. Since SRTP is only implemented protocol, the policy number is fixed and its value is 0. As it is mentioned in section 4.1.1, the CSId value for initiator is 1 as a sender and 2 as a receiver and vice -versa for the responder. Therefore if the target is the initiator, then the CSId value will be 1; while if the target is responder then the CSId value will be 2.
- Next the LEA module generates the Master Key, Master Salt Key, and Authentication Key.
- Then using these keys and the Cryptographic context, the session keys are generated to decrypt each SRTP payload.
- Using the derived session keys, the LEA module decrypts the encrypted SRTP payload of each of the captured SRTP packets (similarly the SRTCP packets can be decrypted).

5.3.2 Algorithm for calculating the ROC

Since ROC is calculated locally by the receiving User Agent and the LEA module works as a receiving User Agent, it needs to calculate the ROC locally. As

stated in the previous subsection, we assume that the full conversation was captured. The algorithm is used to calculate ROC:

```

Initialize ROC to zero (0).

Get sequence number of a captured SRTP packet (sequentially from the
first SRTP packet until the end of the session).

If sequence number!=65535 then return ROC.

Else If (sequence number==65535) {
    Current_ROC=ROC.
    ROC=ROC+1.
    Return Current_ROC.
}

```

5.3.3 Project Description

The LEA module is more or quite similar to the receiving User Agent. The main difference between these two is that the User Agent works on-line and in real-time, while LEA module will work off-line. Additionally, the User Agent locally generates some values using real-time parameters, while the LEA module receives these values from the Escrow Agent. We have used some of the libraries of the existing minisip code for our implementation. Because we want some additional functionality, we have also overloaded some functions of the minisip code. In this implementation we categorize functions into three classes: *readEscrowinfo*, *SetCryptoinfo*, and *Mainclass*. A summary of these three classes are given below.

In the new *readEscrowinfo* class we have place the previously escrowed escrowed parameters that are read from the text file stored previously at the local machine of the Law Enforcement Agency by using the web interface of the LEA module.

In the *SetCryptoinfo* class session keys are generated from the TGK using the escrowed parameters. The resulting session keys are used to decrypt the encrypted payloads of the captured SRTP packet that will be read from the libpcap file. In this class we have added some of the functions to the libminisip library of the existing minisip. Some existing minisip functions are also reused in this class to derive the session keys and to decrypt the packets. The existing classes that are reused are:

```

SrtpPacket::SRtpPacket() [creates SRTP Packet]
SrtpPacket::readpacket(), [Read SRTP packet]
SrtpPacket::protect(), [encrypt the payload of an SRTP packet]
SrtpPacket::unprotect(), [decrypt the payload of an SRTP packet]
SrtpPacket::getBytes(), [retrun total bytes of the SRTP packet]
SrtpPacket::getContent(), [retrun contents of the SRTP packet]
SrtpPacket::size(), [return the size of the SRTP packet]
CryptoContext::CryptoContext(), [Create the Cryptographic context of the SRTP
packets]
CryptoContext::rtp_encrypt(), [encrypts the RTP packet]
CryptoContext::rtp_authenticate(), [authenticate the RTP packet]
CryptoContext::derive_srtp_keys(), [derive SRTP session keys from the TGK]
Certificate::load(), [load the certificate file for the public key]
OsslCertificate::verifSign(), [verify the signed hash]

```

OsslPrivateKey::privateDecrypt() [decrypt the signature with the RSA private key]
 void hmac_sha1(), [generates hash]
 void AES::get_ctr_cipher_stream(), [return encrypted stream of the SRTP packet]
 void AES::ctr_encrypt(), [encrypts data]
 unsigned char * mcrypto::base64_decode(), [decode base64 data]
 Mobject, and CryptoContext. Pseudo random functions void prf(), void p().

We have modified and overloaded the following functions for using off-line working operation in the LEA module:

void KeyAgreement::genAuth(), [generates authentication key]
 void KeyAgreement::genEncr(), [generates encryption key]
 void KeyAgreement::genSalt(), [generates master salt key]
 void KeyAgreement::genTek(), [generates master key]
 void KeyAgreement::keyDeriv(), [derives keys]
 realtimeMediaSender::handleRtpPacket(), [read SSRC, sequence number, generates ROC and process SRTP packet]
 realtimeMediaSender::initCrypto(). [Initialize cryptographic context]

Since minisip works online while LEA module works offline, we have written some functions so that the existing functions (mentioned above) can be re-used.

SrtpPacket::readPacketF1(), [re-packetized a captured libpcap or tcpdump packet as SRTP packet]

SrtpPacket::readPacketF2(), [authenticate a captured libpcap or tcpdump packet as SRTP packet]

RtpPacket::readpacketF1(), [re-packetized and authenticate a captured libpcap or tcpdump packet as an RTP packet]

In the *Mainclass* class packets are read from the captured libpcap file. Third party library libpcap is used for this purpose. In the main class *pcap_open_offline()* and *pcap_loop()* are used to read the packets from libpcap file [60].

The source code for this module is in appendix B and the reused minisip modules are available at www.minisip.org [8].

5.3.4 Capturing a Session

The first step of any electronic surveillance is to capture a communication session between two (or more) communicating peers from the communication channel. In this case, different Law Enforcement Agency uses many different sniffing methods and/or tools. Some times the telecommunication operator or a third party organization performs this job on behalf of the Law Enforcement Agency. To evaluate the proposed TTP based secure Key Escrow method, Wireshark [19] was used to capture a target session. After capturing a full session, we filtered out the SRTP and RTCP packets of the target user and saved these packets on the local machine as a libpcap file.

5.3.5 Operation procedure of the LEA module

After capturing and filtering a session, the saved libpcap file is feed into the LEA module to decrypt its content and to generate an audio file for replay. The whole

procedure was described in detail according to the algorithm depicted in section 5.3 and its visualization form is shown in Figure 5-4.

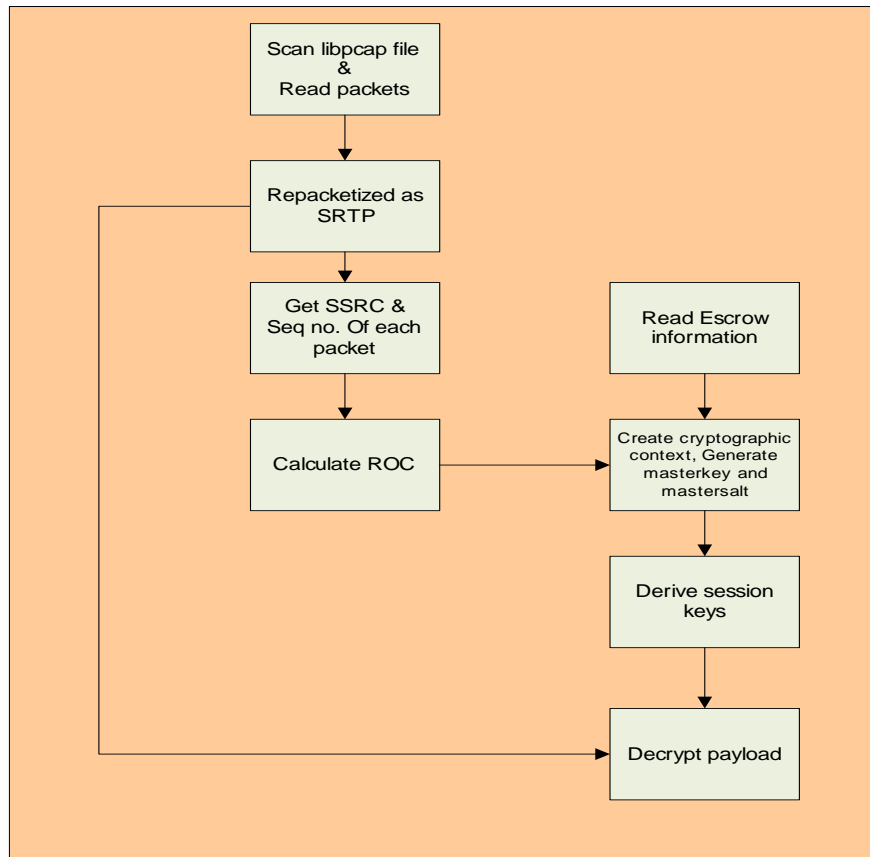


Figure 5-4: Workflow of the Session Key Generation and Payload Decryption

5.4 Validation module

A validation module has been implemented for evaluating the forgery detection mechanism. The algorithm is described in the next subsection, followed by its implementation in the subsequent subsection.

5.4.1 Algorithm for Validation module

1. Read an RTCP packet and stores the signed hash from each RTCP in a vector A.
2. Read the escrowed information.
3. Generate the authentication key using the escrowed information.
4. Check the sequence number of the first SRTP packet.
5. Read SRTP packets and create blocks of n packets.
6. Create a hash of these SRTP packets in the block with the authentication key (generated in step 2).
7. Save these hashes in vector B.
8. Check the last signed hash of A with the escrowed last signed hash.

9. Load public key certificate for the target and create a certificate object.
10. Call the VerifySign function within the OsslCertificate class and pass the hashes (one by one) from vector B to compare with the hashes contain in A (obtained from the signed hashes of the RTCP packets).
11. If the return value is -1, then the certificate file or require a certificate is invalid or missing, if the return value is less than zero (0) we cannot verify the signature, if return value is zero (0), then the data is forged, and if the return value is one (1), then block is valid, i.e., it has not been forged or modified.

5.4.2 Implementation of the Validation Module

We have implemented a validation module to detect all possible types of forgery. In the *verificationRun* class two functions such as: *VerificationRun::run()* and *VerificationRun::runRtcp()* are used to read SRTP and RTCP packet respectively. These two functions use the *pcap_open_offline* function to open the *libpcap* file and *pcap_loop* function to read the *libpcap* file. Both of these functions are from the third party library *libpcap* [60]. The function *rtcpvector (unsigned char *userv variable, const struct pcap_pkthdr * packet header, const u_char *packet)* is used store signatures from the RTCP packets into a vector. Function *setsig::setsig(unsigned char *signature value, unsigned int signature length)* is used to create a vector object.

On the other hand, the function *readblock()* is called with for each SRTP packets. This *readblock()*function creates the SRTP blocks using the *VerificationRun::ProcessBlock (MRef<SRtpPacket *>)*. The function *VerificationRun::CreateHash()*generates authentication keys using the escrowed information and creates a hash of each SRTP block with authentication key. The function *sethash::sethash(unsigned char *hash value, unsigned int hash length)* stores the hash of SRTP blocks into another vector.

For verification, validation module first checksthe sequence number of the first SRTP packet with the sequence number of the first SRTP packet of the escrowed information to check if any forgery or modification has been made in the front of the session. After that, a newly generated hash of the last captured SRTP block is compared with the last signed hash of the escrowed information. This ensures that no forgery has occurred in (or after) the last block. Performing these tests, new hashesare generated from each SRTP block then compared with the signed hash of the captured RTCP blocks sequentially. Another function *verify_hash()* reads the user's public key from his certificate using the *OsslCertificate::load("../*_cert.pem")* function and creates an object with this certificate. With this certificate object the *verifySign* function are called to verify the sent hash (after decrypting the signed hash obtained from the RTCP packet) with the newly generated hash of the captured SRTP blocks fromthe validation module.

5.4.3 Testing Forgery with the Attacker module

In this thesis project, goal of an attacker module is to evaluate the proposed SIP User Agent with Key Escrow [43]. A bad cop in the Law Enforcement Agency can forge a captured session by deleting, inserting, or replacing one or more packets, or by combinations of other operations. Using the attacker module any of these types forgery can be performed. For any modification of an SRTP packet, the checksum value of the UDP header of the each packet must be recalculated with the modified content or payload of each packet. Without re-calculation of the UDP checksum,

forgery would be easily detectable. As shown in the Figure 5-5 wireshark shows the each UDP checksum error. To re-computing the UDP checksum, the algorithm in section 5.4.4 has been used.

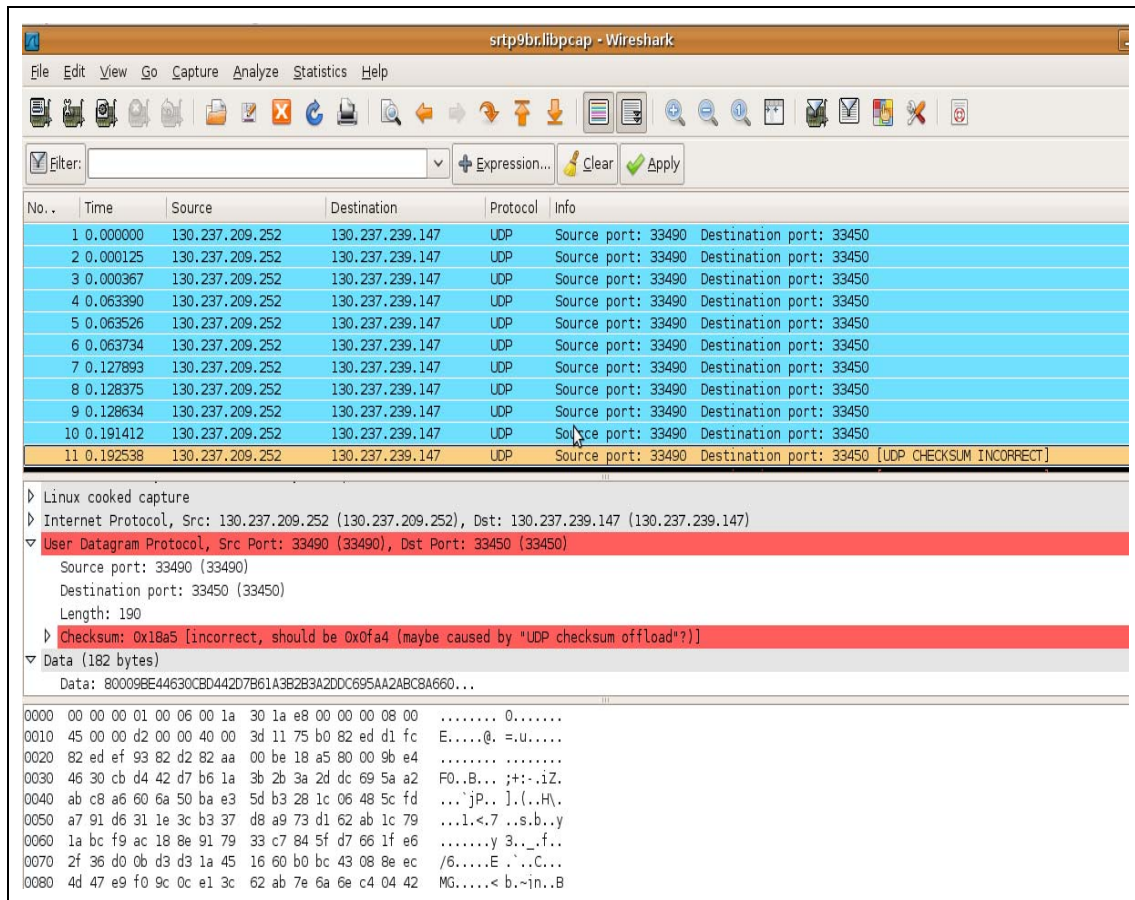


Figure 5-5: Packets with Incorrect UDP Checksum are highlighted by Wireshark

5.4.4 Algorithm for UDP Checksum Calculation

For any successful off-line forgery of a packet, the checksum must be recalculated and inserted into a packet, since without correct checksum value forgery of packets can easily be detected. The checksum of a UDP packet is calculated using octets of the pseudo header, UDP header, and data [55][56]. The pseudo header consists of the IP Source Address (4 bytes), IP Destination Address (4 bytes), Protocol (2 bytes), and UDP length (2 bytes). Additionally, the checksum value in the UDP header must be set to zero before computing the checksum value. The checksum computation is as follows:

1. Read pseudo header of the packet.
2. Read UDP header of the packet.
3. Set checksum byte of the UDP header to zero (0).
4. Read the data octets of the packet.
5. Check the length of the data. If the data octet is odd add a zero padding byte at the end of the packet.
6. Initialize a variable sum =0.

7. Make 16 bit words with two adjacent 8 bit octets of the data.
8. Sum all 16 bit words to create the new checksum.
9. Add the contents of the pseudo header to the sum.
10. Keep only the last 16 bits of the 32 bit calculated sum and add the carries with the sum.
11. Take one's complement of the sum and assign the sum to store the sum in the checksum field.
12. Return checksum.

5.4.5 Working operation of the Attacker Module

The attacker module has been developed in C++ to forge packets in a *libpcap* or *tcpdump* file that containing SRTP packets of a target session. Capturing the conversation is performed by using Wireshark [19]. In this case, a *libpcap* file is captured and saved by using the Wireshark. Before, saving the captured file, it is filtered to contain the SRTP and RTCP packets of the target. Examples of Wireshark filters code for SRTP and RTCP packet are respectively:

```
Ip.src== <ip address> (eg. 130.237.15.252) && rtp && !icmp
```

And

```
Ip.src== <ip address> (eg. 130.237.15.252) && !rtp && !icmp
```

Modification of the *tcpdump* file can be made by inserting, replacing, or adding a packets or block of packets or blocks of packets. In this thesis project, we began by deleting packets or block of packets using the Wireshark. Deleting packets (or groups of packets) are quite easy to do and enable us to quickly check that we could detect these missing packets.

Next we have implemented a module in C++ to forge a *libpcap* file by inserting and/or replacing, and/or modifying a packet, and/or a block of packets, and/or blocks of packets in the front, middle, at the end, or any combination of these. To read the *.libpcap* file, a third party library named “*libpcap*” [60] was used. Functions such as “*pcap_open_offline*”, “*pcap_loop*”, *pcap_dump()*, and *pcap_dump_open()* have been used to deal with the *libpcap* or *tcpdump* file. Purposes of these functions have been depicted in these functions were described in section 2.12. The implemented module first reads a *tcpdump* or *libpcap* file, then modifies it with the various combinations of forgery (more details about the possible forgery described in the section 6.2.1). A summary of the forgery combinations tested using this module is:

1. Delete a packet, a block of packets, or blocks of packets.
2. Insert a fake block that does not have the correct SRTP authentication.
3. Insert a fake block that has the correct SRTP authentication (computed using the secret obtained from the escrow agent).
4. Insert a fake block with the same sequence number as a valid block, but insert this before the block with this sequence number without the correct SRTP authentication.

5. Insert a fake block with the same sequence number as a valid block, but insert this before the block with this sequence number with the correct SRTP authentication.
6. Insert a fake block at the end of the session (after the last valid block in the real recorded session) without the correct SRTP authentication.
7. Insert a fake block at the end of the session (after the last valid block in the real recorded session) with the correct SRTP authentication.
8. Insert a fake block before the start of the session (i.e., before the first valid block in the real recorded session) with the correct SRTP authentication.
9. Insert a fake block before the start of the session (i.e., before the first valid block in the real recorded session) without the correct SRTP authentication.
10. Repeat each of the above, but rather than doing the step with a single SRTP block - use a set of 128 SRTP blocks (Where 128 was be block size used in the original captured session.).

5.4.6 Implementation of the attacker module

In the implementation of the attacker module, we have tested all possible forgery described in section 5.4.5 and in section 6.2.1. Rather than writing separate functions we reuse the same function to test all types of forgery. This was done by modifying a line or few lines of the functions in the C++code (see Appendix C). This allowed us to write seven functions to forge the sessions in different ways. Additionally, three functions are required to read libpcap file, create blocks, and compute UDP checksum:*mainattacker::RunAttacker()*, *rblock ()*, and *compute UDPChecksum()* respectively.

For forgery, we have performed the followings tests:

1. *Forge by removing packets or a Block of packets:* We simply delete a SRTP packet, or a block of SRTP packets, or several blocks of packet from the recorded pcap file using Wireshark.
2. *Forge by inserting packets or a Block of packets:* We have read the tcpdump or pcap file of the target session by using a function *mainattacker::RunAttacker()* and another pcap file (that contains either a SRTP packet, a block of packets, or multiple blocks of packets) that would be used to forge a recorded session. Packets produce by combining these two files will be written to new file. The implemented module can insert any combination of forged packets in any where in the recorded session. To open and write packets into a libpcap file *pcap_dum_open()* and *pcap_dump()* functions from the pcap library have been used.
3. *Forge by replacing packets or a Block of packets or revealing the content of packets:* In this case, the functions *rblock ()*, and *compute UDPChecksum()* are used. Figure 5-6 shows that the data of a SRTP

packet has been replaced and Figure 5-7 shows that some other packets have been inserted without changing the sequence number. Note that to do an actual forgery new time stamps have to be calculated to avoid the huge difference in time between packet number 128 and packet number 129 in Figure 5-7.

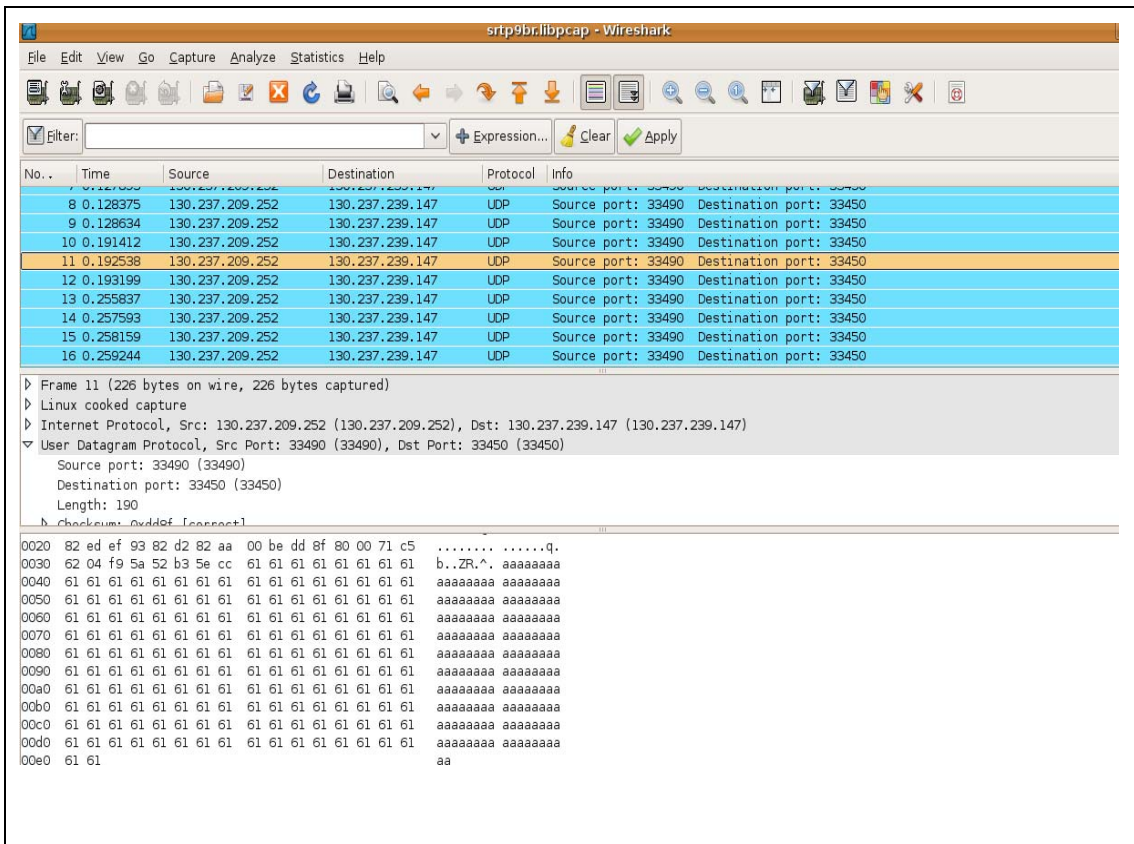


Figure 5-6: A Packet Forged by Replacing the Content

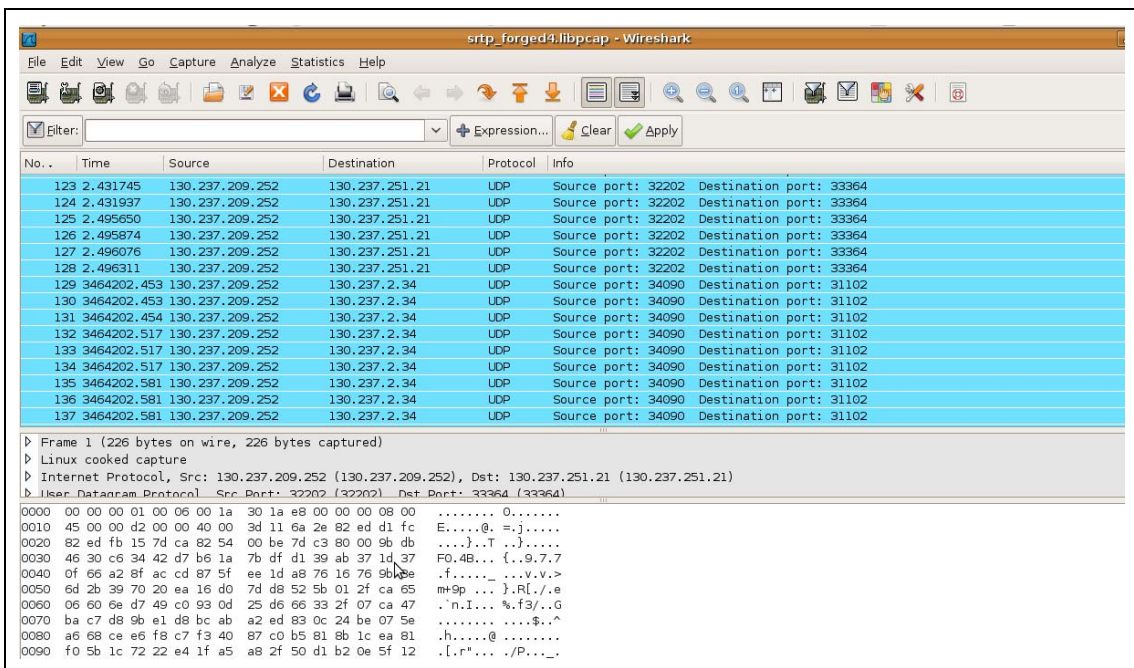


Figure 5-7: Replaced Block by whole Content

Chapter 6: Evaluation of the proposed LI System

Design analysis and implementation of the proposed LI mechanism in the VoIP have been described in previous chapters. This chapter starts with the good cop scenario that shows how a good cop working for a Law Enforcement Agency (LEA) can utilize the proposed approach for intercepting a conversation. Finally, a bad cop scenario depicts showing many different ways that a bad cop could fabricate a recorded session.

6.1 Good cop Scenario

In the first phase of our evaluation, we consider the case of a good cop in a LEA who has captured a real-time session of a target peer using a sniffing tool. In this case, the good cop has captured both SRTP and RTCP packets of full sessions, i.e. from the start of the sessions until end of the sessions. In our implementation we have used the Wireshark [19] to capture a session. Capturing a conversation between two peers will be conducted according to the algorithm in the section 5.3.1. After capturing a session, the good cop asks the Escrow Agent for the TGK and other escrowed information relevant to this session. In this case, we have used a web based interface (web page) for this interaction between the Escrow Agent and the LEA module (as this was shown earlier in the Figure 5-1). Through this web page, an employee of the Law Enforcement Agency will authenticate himself as a valid user of the Escrow Agent module. After validating the LEA User Id another web form will be return to enable the LEA employee to provide the provide target session related information (target User ID and time of the session) and corresponding court warrant. In our implementation (as shown in the Figure 5-2), the interface only requires the target User ID and time of the session. After verification of the target session, the Escrow Agent module returns a page with the TGK and other relevant escrowed information (as was shown in the Figure 5-3). Then God cop can save the escrowed information in a file on the local machine of the Law Enforcement Agency.

For deriving the session keys from the TGK and other escrowed information, LEA module will first read the escrowed information from the text file that previously saved in the local machine by the good cop of Law Enforcement Agency. Using this information LEA module first generates the master key, master salt key, as well as the cryptographic context that is required to derive the session keys and to decrypt the encrypted SRTP payload. In our implementation (as described in section 5.2), the LEA module generates the master key and master salt key as shown in the Figure 6-1 (all values shows in hexadecimal format). Using these keys, session keys are generated and these can be used to decrypt the encrypted payloads as shown in the Figure 6-1. Therefore, using the LEA module, Law Enforcement Agency can generates the session keys for the target session and can decrypt the session to generate an audio file for replay for investigating criminal activities.

```

/root/readwireshark/dist/Debug/GNU-Linux-x86/readwireshark
Master Key:: 38 B3 A2 C0 CE A7 C8 46 5A 5 59 63 66 E9 4A
Master Salt:: E3 CA 34 30 BA 1E 39 AA 26 A6 13 2F 8E C2

Session Encryption Key::
F7 FB 53 D7 3F 1C 6D E1 AA 4C F6 E 14 FA 21 19

Session Authentication Key::
56 5E FB BD 7E 12 83 15 82 3F 2F 1F A1 39 45 C9 7D 62 6E 16

Session Salt Key::
C0 A2 9A AA 15 53 EF 76 C0 9B D3 13 C7 58

Encrypted Payload::
8F 44 59 E6 74 35 97 F0 68 13 37 F0 1E 45 5 AE E2 9C B2 2F E8 F6 71 94 32 56 13
A 15 33 FD 67 D4 98 1B DB 45 95 60 49 D2 59 8E AA D6 6 C3 DA A6 FB F0 38 5C C3 2
B 3F B0 7E B8 E 8B 34 9 44 A6 1C 5F 53 5 1F D3 F8 58 6C 96 9B 46 AD A5 1A A0 29
90 AA 8 55 89 DB E0 3F F3 42 3C 33 7D 4A D5 39 4 38 73 54 3A F9 30 3 66 A0 69 8E
69 B3 C9 EB D3 6D B0 EB 13 BC EE 8E 81 50 D6 84 C0 F3 83 95 C7 F3 71 5B DA 4C 2
D 70 BA 5E CB D8 92 7C 69 70 C7 7A 29 23 14 CD FE 30 33 CC 39 8E 87 EC

Decrypted Payload::
70 BB A6 19 8B 4B 68 F 97 EC C8 F E1 BA FA 51 1D 63 4D D0 17 9 8E 6B CD A9 EC F5
EA CC 2 98 2B E6 E4 24 BA 6A 9F B6 2D A6 71 55 29 F9 3C 25 59 4 F C7 A3 3C D4 C
0 4F 81 47 F1 74 4A F6 BB 59 E3 A0 AC FA E0 2C 7 A7 93 69 64 B9 52 5A E5 5F D6 6
F 55 F7 AA 76 24 1F C0 C BD C3 CC 82 B5 2A C6 FB C7 8C AB C5 6 CF FC 99 5F 96 71
96 4C 36 14 2C 92 4F 14 EC 43 11 71 7E AF 29 7B 3F C 7C 6A 38 C 8E A4 25 B3 D2
8F 45 A1 34 27 6D 83 96 8F 38 85 D6 DC EB 32 1 CF CC 33 C6 71 78 13
Press [Enter] to close the terminal ...

```

Figure 6-1: Session Key Generation and Payload Decryption

After deriving the session keys, the LEA employee may validate the captured session with the validation module whether the session has been fabricated during transmission in the communication channel between two User Agents. The validation module works according to the algorithm as described in the section 5.4.1. For a valid captured session, the user should see a session as shown in the Figure 6-2. This indicates that each of the blocks has been successfully validated.

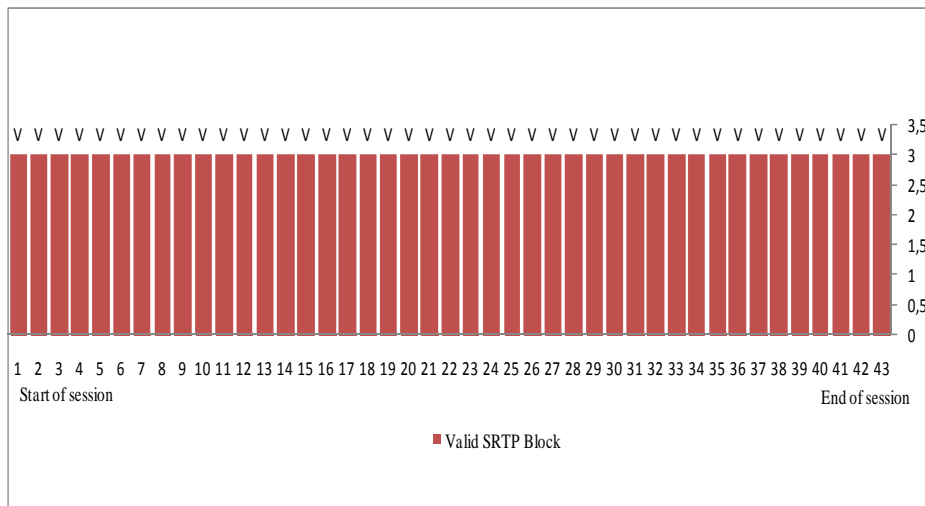


Figure 6-2: Validation output of a Valid SRTP Session

6.1.1 Time required for intercepting a session by the LEA

We have found in our experiment that the average time required for generating the session keys from the TGK and other escrowed information is ~136 micro second (see Figure 6-3). This is a very short time which will have less affects on an off-line session analysis. During evaluation of the performance of the LEA module, we have tested several times with different recorded sessions for finding the time required to derive session keys from the TGK and other escrowed information.

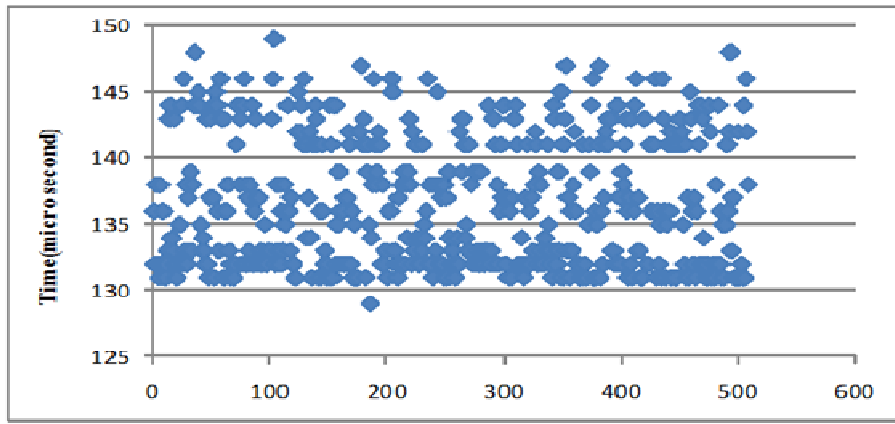


Figure 6-3: Time required generating Session Keys from TGK and other escrowed information

In the statistical analysis of the required time (for details see Appendix E) to derive session keys from the TGK, we have found the following statistical information based on the part of the 1st test as in the Appendix E.

<i>Data analysis on required time for deriving Session keys from TGK and other escrowed information</i>	
Mean	136.5
Median	136
Mode	132
Standard Deviation	4.8
Sample Variance	23
Range	19
Minimum	129
Maximum	149

Here we can see that the lowest time to generate session keys is 129 micro seconds from the TGK, while maximum time required for doing the same thing is 149 micro seconds. Based on the statistical data of the experiment, Figure 6-4 shows the frequency of the SRTP packets and the time to generate session keys.

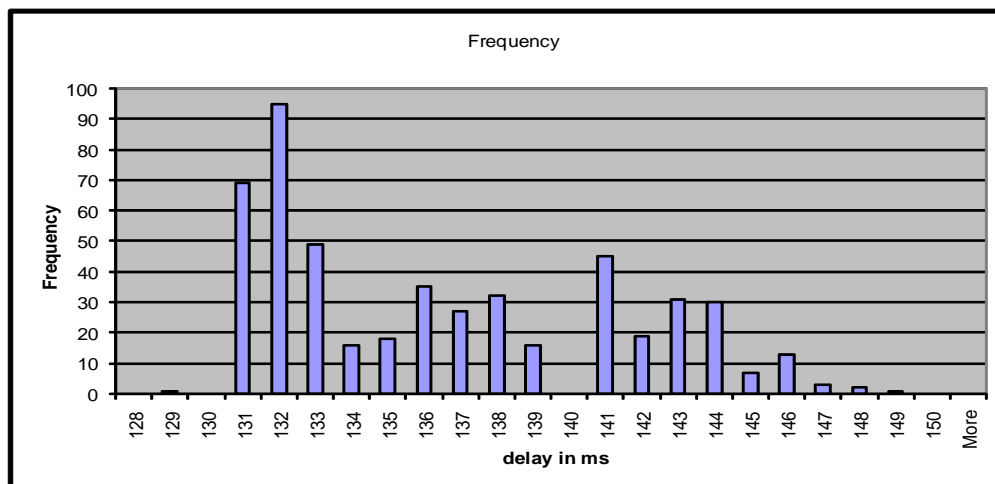


Figure 6-4: Frequency vs. delay for Generating Session Keys

Figure 6-5 shows the cumulative frequency of the delays that beyond the minimum required time 129 microseconds (see Appendix F for statistical data).

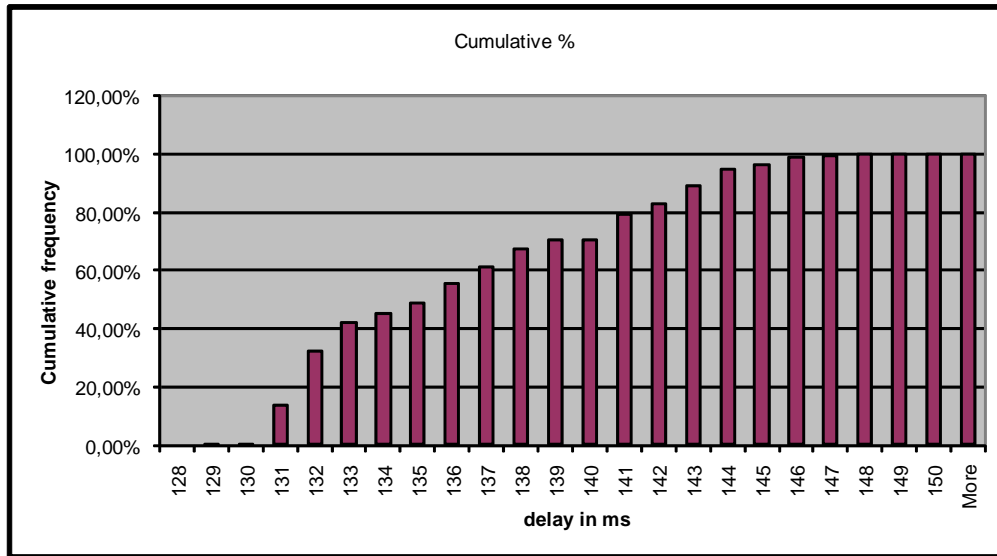


Figure 6-5: Cumulative frequency of the delay

Similar experiment has also been performed to find the time required to decrypt an SRTP packet using the generated session keys. Experiment showed that the mean time required for decrypting a SRTP packet is ~19 micro seconds using the generated session keys. To decrypt the SRTP packets of a recorded session has been shown in the Figure 6-6 (for more details see appendix G).

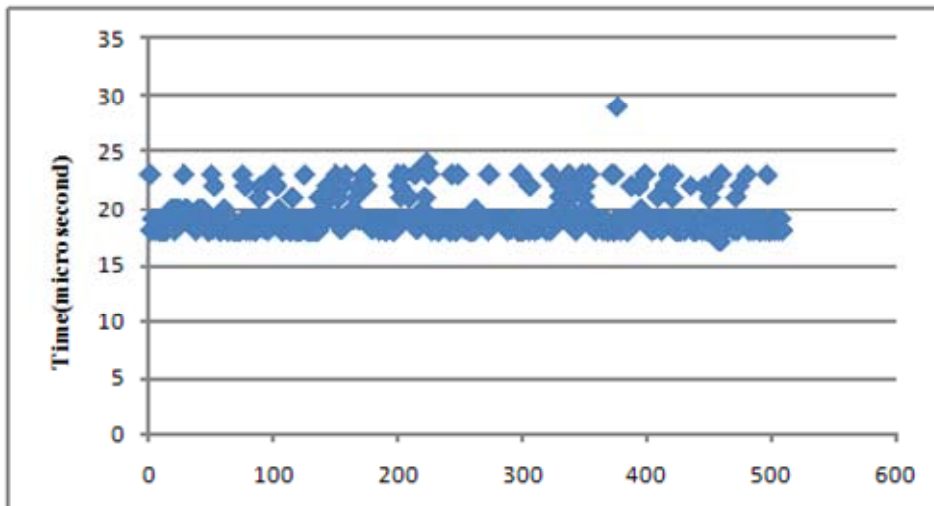


Figure 6-6: Time required to decrypt an SRTP packet using session keys

Based on the experiment for finding out the time required to decrypt SRTP packets, following statistical information has also been measured (for more details see appendix H and I). In this case, the minimum required time is 17 micro seconds while the maximum is 29 micro seconds.

<i>Data analysis on required time for decrypting an SRTP packet</i>	
Mean	19.2
Median	19
Mode	19
Standard Deviation	1.4
Sample Variance	1.96
Range	9
Minimum	17
Maximum	29

Moreover Figure 6-7 shows the frequency of the the time required to decrypt the SRTP packets using the generated session keys, based on the statistical data of the experiment.

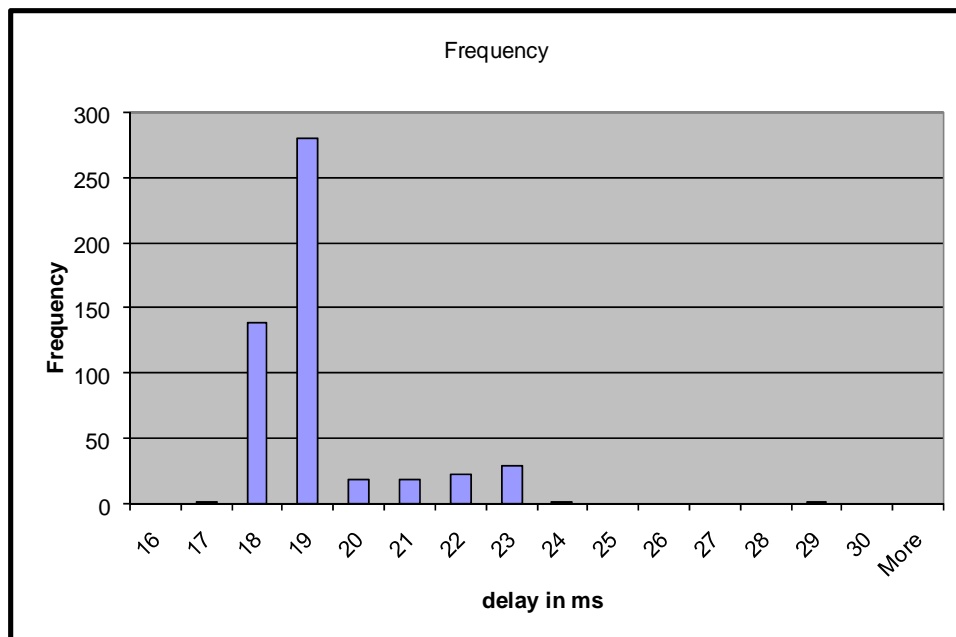


Figure 6-7: Frequency vs. delay for decrypting SRTP packets

Besides, Figure 6-8 shows the cumulative frequency of the delays that beyonds the minimum required time 17 microseconds (see Appendix H for statistical data). As it has been stated in the section 4.2.2.1 that other required delays (such as time for getting court order, getting escrow information from the Escrow Agent, transmission delay, hence the time to generate session keys and then to decrypt a session depends on the specific scenario. For example, to get a court warrant depends on the co-ordination between the judicial department and the crime investigation department of that specific country.

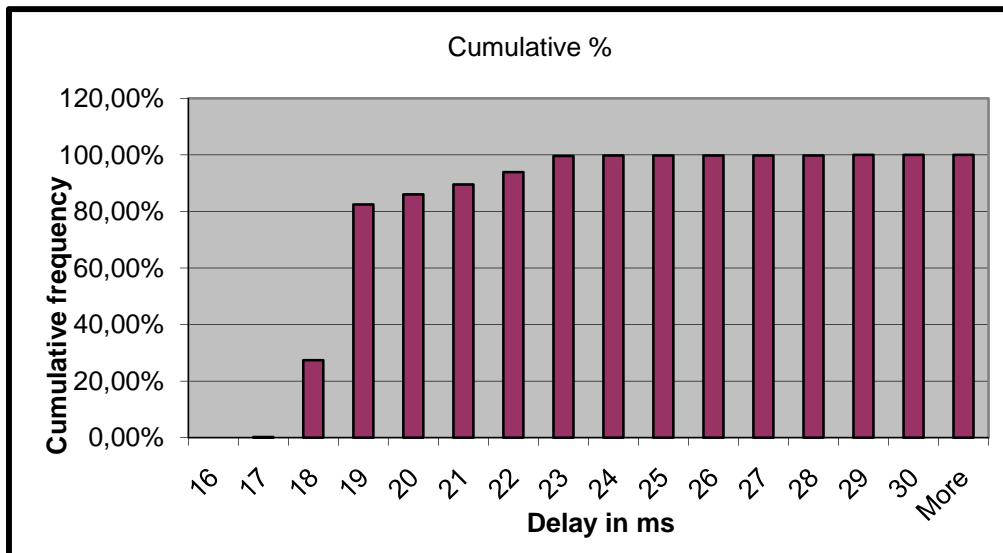


Figure 6-8: Cumulative frequency of the delay for decrypting SRTP packets

6.1.2 A Real-Life Example

Suppose Mr. X is an employee of a Law Enforcement Agency responsible for generating session keys from a TGK and other information as well as decrypting recorded session using the generated session keys. Mr. Y (boss of Mr. X) called Mr. X into his office and gives him a DVD contained with a recorded session of six months worth of calls by a target User Z. Mr. Y tells Mr. X that the sum of the recorded sessions consists of n SRTP packets and m RTCP packets, and asked him to decrypt the session. How long should it take Mr. X to generate a plain-text audio file of the sessions?

From the analysis in section 4.2.2.1 and section 6.1.1, Mr. X can easily calculate the required T_1 time required to generate a session key from the TGK, and the time to decrypt the SRTP payloads with this key. If we ignore the time to compute the session keys (since this only has to be done typically once per session), then if the required time is 155 (136+19) microseconds per SRTP packet, then the time to decrypt the session(s) will be $n * 155$ microseconds once the TGK and other parameters are determined. We can estimate that even if the sessions were continuous for 6 months that the time required to decrypt the session(s) would be less than 33 hours.

To get the TGK Mr. X has to contact the EA and request the escrowed information for the sessions during this period of time. Based upon experience with prior request Mr. X should be able to estimate the amount of time it will take the EA to process the request. If this time is less than the time remaining in the day, then there is a good chance that Mr. Y can have the plain-text version of the capture traffic by the next day.

6.2 Bad Cop Scenario

The aim of this research project was to prevent a bad cop within the Law Enforcement Agency from misusing the key escrow system. Since misuse of the key escrow system is already a controversial issue between users and the governments, a solution for this is required to provide a balanced lawful interception solution. This means that we have to prevent a bad cop from misusing the key escrow system. In this

thesis project we have focused on being able to detect any attempted modification or forgery of a recorded session by a bad cop. In the proposed approach, it is possible to prevent an insider attack from the bad cop as we have shown that we can successfully detect any combination of modifications to the session. To evaluate the proposed Key Escrow approach, we followed Ashby's Law of Requisite Variety, which says variety in a control system must be equal to or larger than the variety of perturbations in order to achieve control [61]; i.e., we must implement more varieties of detection capabilities to identify a forgery that might be done by a bad cop. In this thesis for forgery analysis, we have considered all forgeries/modifications that can be made to a captured session. The proposed approach has been tested against all of these possible forgeries and been shown to detect each of them.

6.2.1 Possible ways of modifying a recorded call

Given a recorded call (as a Wireshark pcap dump file), there are a number of ways that this recording could be modified. In this section we attempt to enumerate each of these possible modifications. Note that any modification represents either some type of error in the recording or an attempt at forgery. We will enumerate these possible modifications in order to be able to show that the method proposed in this thesis can detect each of these modifications. Note that in some cases we cannot distinguish between a single packet being modified, inserted, or deleted and a group of packets being modified, inserted, or deleted – since the signed hashes are computed over blocks; thus in most cases a block is our minimum granularity of detection of a modification. Table 3 enumerates some of the possible modifications of a recorded call that we consider.

Before going into specific types of modifications, we begin with a reminder of how to compute the number of possible combination of n things taken r at a time:

$${}^n C_r = n! / (n - r)! r!$$

First one must consider the size of the modification. The change could either be a single packet being added, a block of packets being added, or multiple blocks being added. The first 3 columns of Table 4 will be referred to as set S_a {a packet, a block, blocks}. Using the formula above we can calculate the number of possible combinations of these three sizes of modifications as the sum of the number of ways of choosing 1 column from 3 (${}^3 C_1 = 3$), plus the number of ways of choosing 2 column from 3 (${}^3 C_2 = 3$), plus the number of ways of choosing 3 column from 3 (${}^3 C_3 = 1$), for a total number of combinations of set S_a is $(3+3+1)=7$.

Next 3 columns of the Table 4 will be referred to as set S_b {insert, replace, delete} The number of ways of choosing 1 column from 3 is ${}^3 C_1 = 3$

The number of ways of choosing 2 column from 3 is ${}^3 C_2 = 3$

The number of ways of choosing 3 column from 3 is ${}^3 C_3 = 1$

Total combination of set S_b is $(3+3+1)=7$

And next 3 columns of the table set S_c {in front, in the middle, at the end}

The number of ways of choosing 1 column from 3 is ${}^3 C_1 = 3$

The number of ways of choosing 2 column from 3 is ${}^3 C_2 = 3$

The number of ways of choosing 3 column from 3 is ${}^3 C_3 = 1$

Total combination of set S_c is $(3+3+1)=7$

Next 2 columns of the table set S_d {payload, whole content}

The number of ways of choosing 1 column from 2 is ${}^2C_1 = 2$

Total combination of set S_d is 2

Next 2 columns of the table set S_e {with sequence number, without sequence number}

The number of ways of choosing 1 column from 2 is ${}^2C_1 = 2$

Total combination of set S_e is 2

Next 2 columns of the table set S_f {with SRTP authentication, without SRTP authentication}

The number of ways of choosing 1 column from 2 is ${}^2C_1 = 2$

Total combination of set S_f is 2

Total combination of the table with the above condition is $(S_a \times S_b \times S_c \times S_d \times S_e \times S_f) = (7 \times 7 \times 7 \times 2 \times 2 \times 2) = 2744$.

On the other hand, Table 4 also shows some examples of the ways of possible forgery that can be made by a bad cop (for more details see appendix A).

Table 3: All Possible Combinations of Forgery/modifications

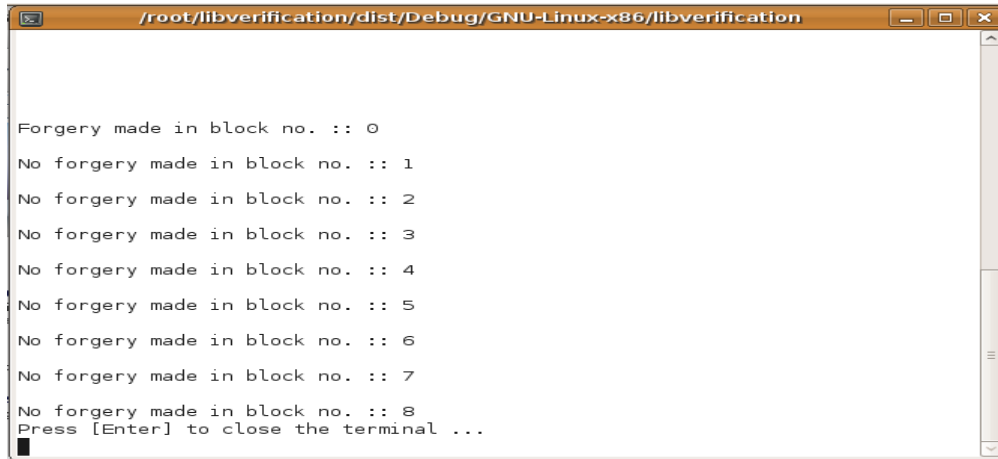
Combination of Forged Options	Number of Combination	Tested	Detectable	Remarks
{a packet, a block, blocks}	7	All	Yes	Which packet is forged is not detectable in the proposed escrow mechanism.
{insert, replace, delete}	7	All	Yes	Without sequence no. of the first SRTP packet, any insertion in the front will show all session as forged.
{front, anywhere in middle, end}	7	All	Yes	Forgery can be made in these three location
{payload, whole packet}	2	All	Yes	Forgery can be made with either payload only or full packet
{with seq. no., without seq. no.}	2	All	Yes	Forgery can be made with either seq. no. or without seq. no.
{with SRTP auth, without SRTP auth}	2	All	Yes	Forgery can be made with either SRTP authentication or without SRTP authentication

Table 4: Examples of Forgery

Forgery Type	A Packet	A Block	Blocks	Insert	Replace	delete	In Front	In The Middle	At The End	Payload Only	Whole Content	With Seq No.	Without Seq No.	With SRTP Auth	Without SRTP Auth
1	X			X			X			X		X		X	
2	X			X			X			X		X			X
3	X			X			X			X			X	X	
4	X			X			X			X			X		X
5	X			X			X				X	X		X	
6	X			X			X				X	X			X
7	X			X			X				X		X	X	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

6.2.2 Detection of the forgery

In the current implementation with the escrow information we build upon the UA that was modified to add signed hashes to the output of a UA and to escrow information with an escrow agent [43], the proposed validation module can detect any of the possible modifications to a recorded session. We have experimentally determined that the verification module can detect all possible forgery of the blocks of using different combinations of types of forgery. Figure 6-9 shows the output when there is a forgery in a block before the start of a session. Figure 6-10 shows the output when there is a forgery in the middle of a session. While Figure 6-11 shows the output when there is an attempt to add a block after the end of a session.



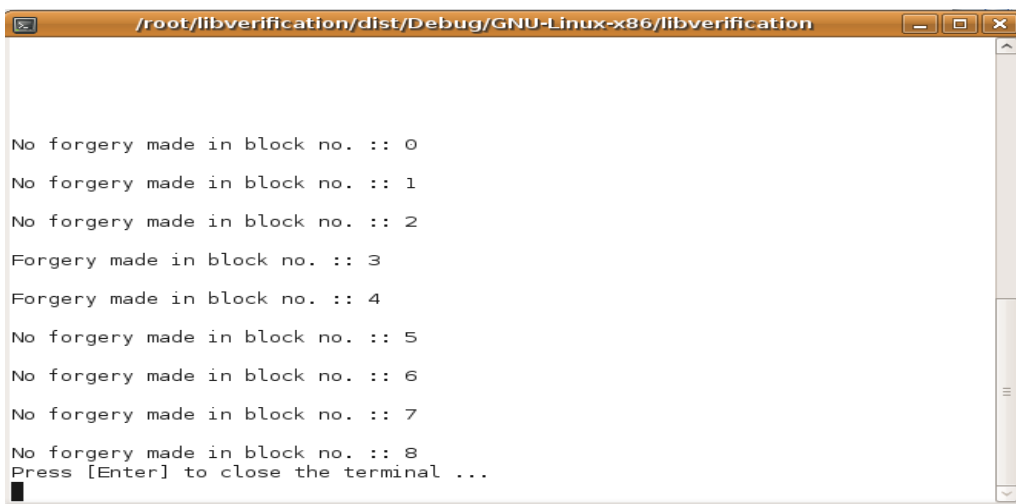
```

/root/libverification/dist/Debug/GNU-Linux-x86/libverification

Forgery made in block no. :: 0
No forgery made in block no. :: 1
No forgery made in block no. :: 2
No forgery made in block no. :: 3
No forgery made in block no. :: 4
No forgery made in block no. :: 5
No forgery made in block no. :: 6
No forgery made in block no. :: 7
No forgery made in block no. :: 8
Press [Enter] to close the terminal ...

```

Figure 6-9: Forged block in the Front of the Session



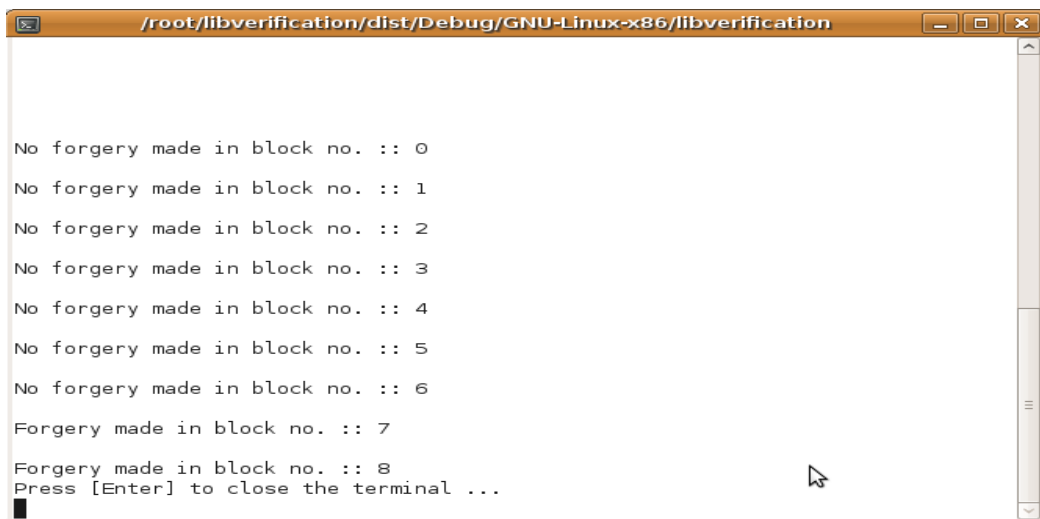
```

/root/libverification/dist/Debug/GNU-Linux-x86/libverification

No forgery made in block no. :: 0
No forgery made in block no. :: 1
No forgery made in block no. :: 2
Forgery made in block no. :: 3
Forgery made in block no. :: 4
No forgery made in block no. :: 5
No forgery made in block no. :: 6
No forgery made in block no. :: 7
No forgery made in block no. :: 8
Press [Enter] to close the terminal ...

```

Figure 6-10: Forged Block in the Middle of the Session



```

/root/libverification/dist/Debug/GNU-Linux-x86/libverification

No forgery made in block no. :: 0
No forgery made in block no. :: 1
No forgery made in block no. :: 2
No forgery made in block no. :: 3
No forgery made in block no. :: 4
No forgery made in block no. :: 5
No forgery made in block no. :: 6
Forgery made in block no. :: 7
Forgery made in block no. :: 8
Press [Enter] to close the terminal ...

```

Figure 6-11: Forged Block at the End of the Session

It is notable that the last block may not be a size of full block, so that it has been dealt specially. When the validation module finds the end of the libpcap or tcpdump file that holds the SRTP packets, it considers the packets that have been collected to

form a block to be a block, which is a hash computed over these packets and compared with the last signed hash. Because the User Agent escrows the last signed hash along with the other escrow information, we can certain that no other blocks or packets have been inserted after the end of the session. Note that as a result of our evaluation of the proposed validation module that we found that the first signed hash also needs to be escrowed to prevent packets from being inserted before the actual session.

During our evaluation, we have fabricated captured sessions using all combinations of forgery using the attacker module (see Table 3 and Table 4). Validation module based on the proposed Key Escrow approach can detect any forgery occurred anywhere in a recorded session. Following Figure 6-12 shows a visualization of this forgery detection of the target session. In this figure we can see that forgery happened in the front, in the middle, as well as at the end of the recorded session. We believe that this visual display of forgery/modifications makes it very easy for even an untrained observer to note that not only has a forgery occurred, but to understand where it has occurred. This type of display could be further improved by showing the relative time offset from the start of the session when a forgery occurs.

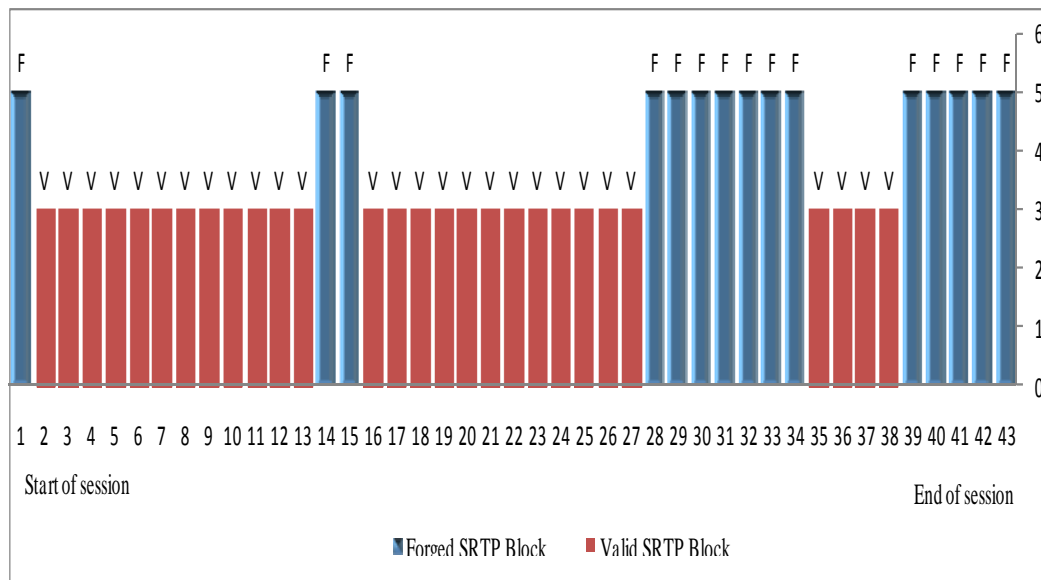


Figure 6-12: Visualized Form of a Forged Session

6.2.3 Shortcomings of the current Escrow Scenario

In our initial investigation for deciding the escrowed information, we have found TGK, CSBID, RAND, and Signed hash (for more details see section 4.1.1) have been required to escrow. Based on this proposal, the proposed LI system with this escrowed information [43] can detect any forged session. But the forged data should be one or more full block size. Otherwise, with this proposed escrow information, it is not possible to detect exactly where the forgery has been happened with the recorded session. If a packets or a number of packets less than a block (or greater than the block size) are inserted with correct sequence number in the front of the session, the proposed solution can only detect forgery/modifications at the granularity of a block. You will find more details about the block size in [43]. It is important to note here that the sender of packets decides upon the size of the blocks that it will sign. Thus to

make the current validation module more automatic, there should be a pre-processing step applied to estimate the block size that was used.

6.2.4 Overcoming this Limitation

In later, we have investigated more about the parameters that should be escrowed with the EA. Additionally, as noted earlier during the work with the validation module we learned that sequence number of the first SRTP packet can be escrowed with the Escrow Agent, since the sequence number is an unpredictable random number, we need to escrow this sequence number or the hash of the first block with the escrow agent. Otherwise modifications could be made before the actual session and these would not be properly detected.

6.2.5 Summary

The goals of this thesis project were:

- Finding the optimum number of parameters that must be required to escrow by the user agent – while allowing the LEA to later generate session keys from the TGK.
- To generate session keys from the TGK along with other escrowed information as needed by the LEA to decrypt the target SRTP session.
- Finding all possible ways to forge a captured SRTP session - as might be used by a bad cop to fabricate or modify the recorded session.
- To prove that the proposed escrow mechanism could detect any forgery occurring with a captured SRTP session.

In this thesis project we have achieved all four goals. In section 4.1.1, we have analyzed and shown the minimum number of parameters that need to be escrowed, along with the reasons for escrowing them. In section 4.2 and section 5.3, we have shown that the session keys can be successfully generated and used to decrypt the captured SRTP session. Section 5.4.5 and section 6.2.1, analyzed and showed all possible forgery combinations illustrating how easy to modify a recorded session. This established that modifying a recording of a session is straight-forward to do and there are several readily available tools for the “Bad Cop”. On the other hand, section 5.4 and section 6.2.2 showed that the proposed key escrow based mechanism can detect any forgery attempt on the recorded session, if the user’s public key and the required escrow information are available. Additionally, we have found that the sequence number of the first SRTP packet must be escrowed in order to detect the exact location of the forgery in a recorded session. Without this sequence number all blocks after the fabricated block will be shown as forged blocks (see section 5.4.3). Hence the sequence number of the first SRTP packet must be escrowed with the Escrow Agent.

Chapter 7: Future Work

Implementing a standard VoIP LI mechanism (that would be acceptable to all relevant parties) is a vast effort. In this thesis, a secure, reliable, and (we hope) acceptable VoIP LI mechanism has been presented. A complete implementation of this mechanism will provide a standard platform that could serve the government purposes while protecting the integrity of the citizens. This will approach prevent an undetectable insider attack which is usually the job of a bad cop (i.e., a bad cop's efforts at forgery or modification would be detected). A pretty basic implementation of this mechanism has been implemented in this thesis project and evaluated (from a performance evaluation purpose point of view). To provide a commercial value and standard would require the following work to be performed in the future:

- In this implementation, a very simple Escrow Agent has been designed. In this implementation the escrowed information is stored in a single database. After getting request from the LEA, the EA retrieves the required information and sends it in a reply to the LEA.

Escrowed information could be disclosed by a dishonest employee of the TTP, i.e., an insider attack. To prevent an insider attack, the EA module needs to be re-designed. In this regard, Adi Shamir's "Secret Sharing" mechanism can be used [62][63]. Using this Secret Sharing mechanism, the secret is split into N shares and distribute to N shareholders. To reconstruct the secret, at least $(t+1)$ shareholders need to agree to disclose their share. Here t is the threshold value. In a similar fashion, we can split the escrowed information and store these share in a number of EAs. Using this mechanism makes an insider attack increasingly difficult as $t+1$ employee of these different EAs have to be dishonest. Abdullah Afzar is examining this in his thesis project [64].

- Certificate handling along with the checking of a certificate revocation list (CRL) during the communication between UA and EA should be added. In addition, this check should occur in the EA and LEA modules. In this implementation, self-signed certificates were used, while in a commercial implementation these certificates should be signed by a recognized CA.
- Automated session key derivation from the escrowed information without transferring them in a human readable form should be added wto the LEA module; i.e. when the LEA module requests for the escrow information to derive the session keys, escrow information will be transferred and used in the back end without saving in the local directory. The transfer from the EA to the LEA module should be protected to reduce the risk of an implementation would keep the LEA module free from insider attack.
- The EA may release records of escrowed sessions either in a batch or in a stream, depending upon whether the ending time is before or after the current time. In case of a stream of records the court order may specify a bounded delivery time. In this current implementation, batch

delivery of records has been implemented. Providing a stream of escrowed information to a LEA should be implemented in the future.

- Setting up a convenient time and secure means for the LEA to get a court order and to send it provide this court order to the EA should be implemented. And implementation of this time frame in this proposed VoIP Lawful Intercept mechanism will be done in future. This requires that a means for the EA to check the validity of the court order should also be implemented.
- A complete security analysis of the LEA module along with the secure log of the LEA module's operations should be performed in the future. This is important both to provide a chain of custody of the evidence and to prevent insider misuse.

Chapter 8: Conclusions

Preventing terrorism and investigation of crime have become global issues today. Since the interception of IP telephony is significantly more complex than for traditional telephony; and because it is easy to encrypt digital traffic –the use of VoIP has become popular among criminals. Moreover, telephone interception or wiretapping is seen by citizens in many countries as a violation of an individual's privacy. Additionally, misuse of electronic surveillance by LEAs has also led to disputes about the lawful interception. All of these have contributed to increased efforts to constrain the use of interception. For this reason, a standard lawful interception mechanism is badly needed so that LEAs can fulfill their need for electronic surveillance while preventing misuse of this capability.

One of the solutions that has been proposed to facilitate LI, while protecting individuals' privacy is key escrow. However, there have been many concerns about the security and integrity of key escrow schemes. This leads to the proposal for combining key escrow with a signed hashing scheme to make the forgery of "captured traffic" detectable.

In this thesis project we have designed, implemented, and evaluated a LI System that performs the above functions. To support this model, we developed four modules: an Escrow Agent (EA) module, a LEA module, a verification module, and an attacker module. The first three modules are the main components of the proposed LI model. While the final module enables us to verify that a captured SRTP + SRTCP session has not been tampered with.

During our evaluation of the proposed LI system we have examined a number of metrics and found that the the system is able to detect any kind of forgery of the conversation session made by the bad cop or by others. We have also found that the time required for deriving session keys and decrypting the recorded session requires is very reasonable,hence this method is quite feasible for an off-line data analysis.

In this thesis project we have developed a simple LI system that was evaluated functionally for performance analysis. However, the effort required to standardize this system for commercial use or for practical implementation out of the scope of this thesis project. Hence it remains for future work.

References

- [1] Dorothy E. Denning, Dennis K. Branstad, “A Taxonomy for key Escrow Encryption Systems”, *Communication of the ACM*, Vol. 39, No. 3, March 1996.
- [2] Philip A. Branch, “Lawful Interception of the Law”, CAIA Technical report 030606A, 06 June 2003. <http://caia.swin.edu.au/reports/>. Last Accessed on 08-09-2009.
- [3] Jack Brooks, “Communications Assistance for Law Enforcement Act (CALEA)”, Committee on the Judiciary, U.S. Congress, <http://www.askcalea.net/>. October 1994. Last Accessed on 21-09-2009.
- [4] Cisco System Inc, “Lawful intercept Architecture”, 2007, <http://www.cisco.com/>, Last Accessed on 11-09-2009.
- [5] Telecommunication Standardization Policy Division, ITU-Telecommunication Standardization Sector, “Technical aspects of lawful interception”, ITU-T technical watch report # 6, May 2008.
- [6] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data, *Official Journal of the European Communities of 23 November 1995 No L 281 p. 31*. http://www.cdt.org/privacy/eudirective/EU_Directive_.html, last accessed on 12-09-2009.
- [7] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman, “MIKEY: Multimedia Internet KEYing”, IETF, Network Working Group, RFC 3830, August 2004. <http://www.ietf.org/rfc/rfc3830.txt> last Accessed on 13-09-2009.
- [8] Minisip, <http://www.Minisip.org/>, last accessed on 28-09-2009.
- [9] Electronic Privacy information Center, The Clipper Chip, “<http://epic.org/crypto/clipper/>”, last accessed on 29-09-2009.
- [10] SKIPJACK and KEA Algorithm Specifications, “<http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf>”, 29 May 1998, Last Accessed on 29-09-2009.
- [11] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman “The Secure Real-Time Transport Protocol (SRTP)”, IETF, Network Working Group, RFC3711, March 2004. <http://www.ietf.org/rfc/rfc3711.txt>
- [12] Adaptive Digital Technologies Inc. “Secure Real Time transport Protocol, Protocol Stack” http://www.adaptivedigital.com/product/protocol_stacks/srtp.htm Last accessed on 08-10-2009.
- [13] T. Friedman, R. Caceres, and A. Clark (Editors), “RTP Control Protocol Extended Reports (RTCP XR)”, IETF, Network Working Group, RFC 3611, November 2003, <http://www.ietf.org/rfc/rfc3611.txt>
- [14] H. Schulzrinne, et al., “RTP: A Transport Protocol for Real-Time Applications”, RFC 3550, Network Working Group, <http://www.armware.dk/RFC/rfc/rfc3550.html>. Last Accessed on 29-09-2009.
- [15] Office of the Inspector General, “The implementation of the Communications Assistance For law Enforcement Act”, U.S. Department of Justice, Audit Division, Audit report, 6-13 March 2006.
- [16] Romanidis Evripidis, “Lawful Interception and Countermeasures: In the era of Internet Telephony”, Master’s Thesis, School of Information and Communication Technology, Royal Institute of Technology (KTH), Sweden,

- COS/CCS 2008-20, September 2008. http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080922-Romanidis_Evripidis-with-cover.pdf
- [17] Erik Eliasson, "Secure Internet Telephony: Design, Implementation, and Performance Measurements", Licentiate of Technology thesis, TRITA-ICT/ECS AVH 06:04, May 2006, Telecommunication Systems Laboratory Electronic, Computer and Software Systems, Royal Institute of Technology (KTH), Stockholm, Sweden. http://www.minisip.org/publications/ErikEliasson_LicentiateThesis.pdf
- [18] Foreign Intelligence Surveillance Act (FISA), Electronic Privacy Information Center, Department of Justice, USA, <http://epic.org/>. Last accessed on 30-09-2009.
- [19] Wireshark, <http://www.wireshark.org/>. Last accessed on 26-09-2009.
- [20] Mikael Svensson, Countering VoIP Spam: Up-Cross-Down Certificate Validation, Master thesis, KTH, Stockholm, September 2007. http://www.minisip.org/publications/Thesis_Svensson_Sep2007.pdf
- [21] E. Rescorla, "Diffie-Hellman Key Agreement Method", RFC 2631, Network Working Group, [http://www.ietf.org/rfc/rfc2631.txt\(diffie-hellman\)](http://www.ietf.org/rfc/rfc2631.txt(diffie-hellman)). Last accessed on 23-09-2009.
- [22] J. Callas, "Open PGP Message format", IETF, Network Working Group, RFC4880, November 2007. <http://tools.ietf.org/html/rfc4880>. Last Accessed on 18-09-2009.
- [23] T. Dierks, and E. Rescorla, "The Transport Layer Security (TLS) Protocol", RFC5246, Network Working Group, August 2008. <http://www.faqs.org/rfcs/rfc5246.html>, Last accessed on 22-09-2009.
- [24] D. Atkins et al., "PGP Message Exchange Formats", RFC 1991, Network Working Group, August 1996, <http://www.ietf.org/rfc/rfc1991.txt>. Last accessed on 22-09-2009.
- [25] M. Euchner, "HMAC-Authenticated Diffie-Hellman for Multimedia Internet KEYing (MIKEY)", RFC4650, September 2006. <http://www.ietf.org/rfc/rfc4650.txt>. Last Accessed on 18-09-2009.
- [26] Anoop MS, "Public Key Cryptography-Applications Algorithms and Mathematical Explanations", <http://www.infosecwriters.com>. Last accessed on 24-09-2009.
- [27] D. Ignjatic, L. Dondeti, F. Audet, and P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", IETF Network Working Group, RFC 4738, November 2006. <http://www.ietf.org/rfc/rfc4738.txt>. Last accessed on 26-09-2009.
- [28] Theodor W. Schlickmann, "Ensuring trust and security in electronic communication", Commission of the European Communities, Directorate-General XIII - Telecommunications, Information Market and Exploitation of Research, Security of telecommunications and information systems. http://www.oss.net/dynamaster/file_archive/040319/e12138381ec03c1c6012940f8d0a3136/OSS1998-E1-08.pdf, Last accessed on 28-09-2009.
- [29] F. Baker, B. Foster, and C. Sharp, "Cisco Architecture for Lawful Intercept in IP Networks", IETF, Network Working Group, RFC 3924, October 2004, <http://www.ietf.org/rfc/rfc3924.txt>, Last accessed on 30-09-2009.
- [30] Joel Weise, "Public Key Infrastructure Overview", SunPSSM Global Security Practice, Sun BluePrints™ OnLine, Sun Microsystems Inc, August 2001.
- [31] N. Kapidzic and A. Davidson, "A Certificate Management System: structure, functions and protocols", Proceedings of the 1995 Symposium on Network

- and Distributed System Security (SNDSS'95), IEEE Computer Society, Washington, DC, USA, 1995, page: 153.
- [32] R. Housley, W. Ford, W. Polk, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 2459, January 1999. <http://www.ietf.org/rfc/rfc2459.txt>, Last accessed on 27-09-2009.
- [33] C. Hett, N. Kuntze, A.U.Schmidt, "Non-repudiation of Coice-over-IP", Fraunhofer SIT, Darmstadt, Germany. Last Access on 05-10-2009
- [34] Erland Jonsson, "KEY ESCROW – a System for Law-Enforced Covert Surveillance and its Risks", Department of Computer Engineering, Chalmers University of Technology, 1 December 2004.
- [35] J. Rosenberg et al., "SIP: Session Initiation Protocol", IETF, Network Working Group, RFC 3261, June 2002, <http://www.ietf.org/rfc/rfc3261.txt>, Last accessed on 07-09-2009.
- [36] SIP center "Understanding SIP", Sip Center, "www.sipcenter.com/sip.nsf/WEBB5YNVK8/\$FILE/Ubiquity_SIP_Overview.pdf", Last accessed on 06-09-2009.
- [37] National Institute of Standards and Technology (NIST), "Escrowed Encryption Standard (EES)", FIPS Publication 185, February 1994.
- [38] Frank W. Sudia, "Private Key Escrow System", overheads of presentation, Bankers Trust Co., New York, NY, 1995.
- [39] Mihir Bellare, and Shafi Goldwasser, "Variable Partial Key Escrow", University of California at San Diego, CSE Department, Technical Report CS95-447.
- [40] Mihir Bellare and Shafi Goldwasser, "Variable Cryptographic Time Capsule: A new Approach to Key Escrow", Manuscript, April 1996.
- [41] Thomas Beth, Hans-Joachim Knobloch, Marcus Otten, Gustavus J. Simmons, and Peer Wichman, "Clipper Repair kit – Towards Key Escrow Systems", Proceedings of 2nd ACM Conference on Communication and Computer Security, 1994.
- [42] David P. Maher, "Crypto Backup and Key Escrow", Communications of the ACM, Volume 39, Issue 3, Pages: 48 – 53, 1996.
- [43] Md. Sakhawat Hossen, A session Initiation Protocol User Agent with Key Escrow: providing authenticity for recording session, Masters Thesis, School of Information and Communication Technology, Royal Institute of Technology (KTH), Stockholm, Sweden, January 2010.
- [44] "Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data", Official Journal of the European Communities of 23 November 1995 No L. 281 p. 31, <http://www.cdt.org/privacy/eudirective/EUDirective.html>, Last accessed on 07-09-2009.
- [45] "White Paper – Lawful Intercept overview", Newport Networks, <http://networks.com/whitepapers/lawful-intercept1.html>, Last Accessed on 04-09-2009.
- [46] D. Ma, M. Abe, and V. D. Gligor, "Practical forward secure sequential aggregate signatures", In proceedings of the ACM Symposium on Information, Computer and Communication Security, ACM, 2008, pp. 341-352.
- [47] Rafael Accorsi, "Log Data as Digital Evidence: What Secure Logging Protocols Have to Offer?", 2009 33rd Annual IEEE International Computer

- Software and Applications Conference, Seattle, Washington, USA, July 20-July 24, 2009.
- [48] Vassilios Stathopoulos, Panayiotis Kotzanikolaou, and Emmanouil Magkos, "A Framework for Secure and Verifiable Logging in Public Communication Networks", Critical Information Infrastructures Security, Volume 4347/2006, Springer Berlin/Heidelberg, 2006.
- [49] D. Ignjatic et al., "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", RFC 4738, IETF, November 2006.
- [50] J. Bilién, "Key Agreement for secure Voice over IP", Master Thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, IMIT/LCN 2003-14, December 2003. <http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/031215-Johan-Bilien-report-final-with-cover.pdf>
- [51] Elisabetta Carrara, "Security for IP Multimedia Applications over Heterogeneous Networks", Licentiate thesis, School of Information and Microelectronics Technology, Royal Institute of Technology (KTH), Sweden, 31 August 2004. <http://web.it.kth.se/~carrara/lic.pdf>
- [52] Shana K. Rahavy, "The Federal Wiretap Act: the Permissible Scope of Eavesdropping in the Family Home", The Journal of High Technology Law, vol. II, No 1, 2003, pages 95.-98.
- [53] Henry Sinnreich and Alan B. Johnston, [Internet Communications Using SIP: Delivering VoIP and Multimedia Services with Session Initiation Protocol](#), 2nd Edition, Wiley, August 2006, ISBN: 0-471-77657-2.
- [54] S. Kent, "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management", IETF, Network Working Group, RFC 1422, February 1993, <http://www.networksorcery.com/enp/rfc/rfc1422.txt>, Last Accessed on 22-10-2009.
- [55] T. Berners-Lee, R. Fielding, U. C. Irvine, and L. Masinter, "Uniform Resource identifiers (URI)", IETF, Network Working Group, RFC 2396, August 1998, <http://www.ietf.org/rfc/rfc2396.txt>, Last Accessed on 23-10-2009.
- [56] J. Postel, "User Datagram Protocol", RFC 768, IETF, August 1980.
- [57] <http://www.cox.com/Policy/leainformation/CoxLawfulInterceptWorksheet.pdf>, Last Accessed on 29-12-2009.
- [58] <http://msdn.microsoft.com/en-us/library/aa380513> (VS.85).aspx. Last Accessed on 30-12-2009.
- [59] http://en.wikipedia.org/wiki/Transmission_delay, Last Accessed on 30-12-2009.
- [60] Libpcap, Beyond Linux From Scratch-Version 6.3, Chapter 14, Networking Libraries, <http://www.linuxfromscratch.org/blfs/view/6.3/basicnet/libpcap.html>, Last Accessed on 11-11-2009.
- [61] W. Ross Ashby, "An Introduction to Cybernetics", Champan & Hall, <http://pespmc1.vub.ac.be/ASHBBOOK.html>, Last Access Date 29-11-2010.
- [62] Shamir's Secret Sharing, "http://en.wikipedia.org/wiki/Shamir's_Secret_Sharing#Mathematical_definition", Last Accessed on 11-12-2009.
- [63] RSA Laboratories, "<http://www.rsa.com/RSALABS/node.asp?id=2259>", Last Accessed on 11-12-2009.
- [64] Abdullah Azfar, "Multiple Escrow Agent in VoIP", Masters Thesis, School of Information and Communication Technology, Royal Institute of Technology (KTH), Stockholm, Sweden, June 2010 (expected).

Appendices

A. Different Forgery Combination

Forgery Type	A Packet	A Block	Blocks	Insert	Replace	delete	In Front	In The Middle	At The End	Payload Only	Whole Content	With Seq No.	Without Seq No.	With SRTP Auth	Without SRTP Auth
1	X			X			X			X		X		X	
2	X			X			X			X		X			X
3	X			X			X			X			X	X	
4	X			X			X			X			X		X
5	X			X			X				X	X		X	
6	X			X			X				X	X			X
7	X			X			X				X		X	X	
8	X			X			X				X		X		X
9	X				X		X			X		X		X	
10	X				X		X			X		X			X
11	X				X		X			X			X	X	
12	X				X		X			X			X		X
13	X				X		X				X	X		X	
14	X				X		X				X	X			X
15	X				X		X				X		X	X	
16	X				X		X				X		X		X
17	X					X	X			X		X		X	
18	X					X	X			X		X			X
19	X					X	X			X			X	X	
20	X					X	X			X			X		X

Appendices

21	X					X	X				X	X		X	
22	X					X	X				X	X			X
23	X					X	X				X		X	X	
24	X					X	X				X		X		X
25	X			X				X		X		X		X	
26	X			X				X		X		X			X
27	X			X				X		X			X	X	
28	X			X				X		X			X		X
29	X			X				X			X	X		X	
30	X			X				X			X	X			X
31	X			X				X			X		X	X	
32	X			X				X			X		X		X
33	X				X			X		X		X		X	
34	X				X			X		X		X			X
35	X				X			X		X			X	X	
36	X				X			X		X			X		X
37	X				X			X			X	X		X	
38	X				X			X			X	X			X
39	X				X			X			X		X	X	
40	X				X			X			X		X		X
41	X					X		X		X		X		X	
42	X					X		X		X		X			X
43	X					X		X		X			X	X	
44	X					X		X		X			X		X
45	X					X		X			X	X		X	
46	X					X		X			X	X			X
47	X					X		X			X		X	X	

Appendices

48	X					X		X			X		X		X
49	X			X					X	X		X		X	
50	X			X					X	X		X			X
51	X			X					X	X			X	X	
52	X			X					X	X			X		X
53	X			X					X		X	X		X	
54	X			X					X		X	X			X
55	X			X					X		X		X	X	
56	X			X					X		X		X		X
57	X				X				X	X		X		X	
58	X				X				X	X		X			X
59	X				X				X	X			X	X	
60	X				X				X	X			X		X
61	X				X				X		X	X		X	
62	X				X				X		X	X			X
63	X				X				X		X		X	X	
64	X				X				X		X		X		X
65	X					X			X	X		X		X	
66	X					X			X	X		X			X
67	X					X			X	X			X	X	
68	X					X			X	X			X		X
69	X					X			X		X	X		X	
70	X					X			X		X	X			X
71	X					X			X		X		X	X	
72	X					X			X		X		X		X
73		X		X			X			X		X		X	
74		X		X			X			X		X			X

Appendices

75		X		X			X			X			X	X	
76		X		X			X			X			X		X
77		X		X			X			X	X		X		
78		X		X			X			X	X				X
79		X		X			X			X			X	X	
80		X		X			X			X			X		X
81		X			X		X			X		X		X	
82		X			X		X			X		X			X
83		X			X		X			X			X	X	
84		X			X		X			X			X		X
85		X			X		X			X	X		X		
86		X			X		X			X	X				X
87		X			X		X			X			X	X	
88		X			X		X			X			X		X
89		X				X	X			X		X		X	
90		X				X	X			X		X			X
91		X				X	X			X			X	X	
92		X				X	X			X			X		X
93		X				X	X			X	X		X		
94		X				X	X			X	X				X
95		X				X	X			X			X	X	
97		X				X	X			X			X		X
98		X		X				X		X		X		X	
99		X		X				X		X		X			X
100		X		X				X		X			X	X	
101		X		X				X		X			X		X
102		X		X				X		X	X		X		

Appendices

103		X		X				X			X	X			X
104		X		X				X			X		X	X	
105		X		X				X			X		X		X
106		X			X			X		X		X		X	
107		X			X			X		X		X			X
108		X			X			X		X			X	X	
109		X			X			X		X			X		X
110		X			X			X			X	X		X	
111		X			X			X			X	X			X
112		X			X			X			X		X	X	
113		X			X			X			X		X		X
114		X				X		X		X		X		X	
115		X				X		X		X		X			X
116		X				X		X		X			X	X	
117		X				X		X		X			X		X
118		X				X		X			X	X		X	
119		X				X		X			X	X			X
120		X				X		X			X		X	X	
121		X				X		X			X		X		X
122		X		X					X	X		X		X	
123		X		X					X	X		X			X
124		X		X					X	X			X	X	
125		X		X					X	X			X		X
126		X		X					X		X	X		X	
127		X		X					X		X	X			X
128		X		X					X		X		X	X	
129		X		X					X		X		X		X

Appendices

130		X			X				X	X		X		X	
131		X			X				X	X		X			X
132		X			X				X	X			X	X	
133		X			X				X	X			X		X
134		X			X				X		X	X		X	
135		X			X				X		X	X			X
136		X			X				X		X		X	X	
137		X			X				X		X		X		X
138		X				X			X	X		X		X	
139		X				X			X	X		X			X
140		X				X			X	X			X	X	
141		X				X			X	X			X		X
142		X				X			X		X	X		X	
143		X				X			X		X	X			X
144		X				X			X		X		X	X	
145		X				X			X		X		X		X
146			X	X			X			X		X		X	
147			X	X			X			X		X			X
148			X	X			X			X			X	X	
149			X	X			X			X			X		X
150			X	X			X				X	X		X	
151			X	X			X				X	X			X
152			X	X			X				X		X	X	
153			X	X			X				X		X		X
154			X		X		X			X		X		X	
155			X		X		X			X		X			X
156			X		X		X			X			X	X	

Appendices

157			X		X		X			X			X		X
158			X		X		X			X	X		X		
159			X		X		X			X	X				X
160			X		X		X			X		X	X		
161			X		X		X			X		X			X
162			X			X	X			X		X		X	
163			X			X	X			X		X			X
164			X			X	X			X			X	X	
165			X			X	X			X			X		X
166			X			X	X			X	X		X		
167			X			X	X			X	X				X
168			X			X	X			X		X	X		
169			X			X	X			X		X			X
170			X	X				X		X		X		X	
171			X	X				X		X		X			X
172			X	X				X		X			X	X	
173			X	X				X		X			X		X
174			X	X				X		X	X		X		
175			X	X				X		X	X				X
176			X	X				X		X		X	X		
177			X	X				X		X		X			X
178			X		X			X		X		X		X	
179			X		X			X		X		X			X
180			X		X			X		X			X	X	
181			X		X			X		X			X		X
182			X		X			X		X	X		X		
183			X		X			X		X	X				X

Appendices

184			X		X			X			X		X	X	
185			X		X			X			X		X		X
186			X			X		X			X			X	
187			X			X		X			X				X
188			X			X		X					X	X	
189			X			X		X					X		X
190			X			X				X	X			X	
191			X			X				X	X				X
192			X			X				X			X	X	
193			X			X				X			X		X
194			X	X					X	X			X		X
195			X	X					X	X			X		X
196			X	X					X	X			X	X	
197			X	X					X	X			X		X
198			X	X					X		X	X		X	
199			X	X					X		X	X			X
200			X	X					X		X		X	X	
201			X	X					X		X		X		X
202			X		X				X	X			X		X
203			X		X				X	X			X		X
204			X		X				X	X			X	X	
205			X		X				X	X			X		X
207			X		X				X		X	X		X	
208			X		X				X		X	X			X
209			X		X				X		X		X	X	
210			X		X				X		X		X		X
211			X			X			X	X			X		X

Appendices

212			X			X			X	X		X			X
213			X			X			X	X			X	X	
214			X			X			X	X			X		X
215			X			X			X		X	X		X	
216			X			X			X		X	X			X
217			X			X			X		X		X	X	
218			X			X			X		X		X		X

B. Source code of LEA Module

```

/*
 * File: setcryptoinfo.cpp
 * Author: morhsed
 *
 * Created on November 29, 2009, 11:12 PM
 */
#include "setCryptoinfo.h"
#include</root/readwireshark/derivekeys.h>
#include<libminisip/media/MediaStream.h>
#include<libmikey/MikeyPayloadSP.h>
#include<string.h>
#include<stdlib.h>

#ifdef _WIN32_WCE
# include"../include/minisip_wce_extra_includes.h"
#endif

#ifdef SCSIM_SUPPORT
#include<libmcrypto/SipSimSmartCardGD.h>
#endif

static byte_t ipsec4values[] = {MIKEY_IPSEC_SATYPE_ESP,
MIKEY_IPSEC_MODE_TRANSPORT,MIKEY_IPSEC_SAFLAG_PSEQ,MIKEY_
IPSEC_EALG_3DESCBC,24,MIKEY_IPSEC_AALG_SHA1HMAC,16};

static byte_t srtpvalues[]={MIKEY_SRTP_EALG_AESCM,16,
MIKEY_SRTP_AALG_SHA1HMAC,20,14,MIKEY_SRTP_PRF_AESCM,0,1,1,MI
KEY_FEC_ORDER_FEC_SRTP,1,10,0};

using namespace std;

/* serves as define to split inkey in 256 bit chunks */
#define PRF_KEY_CHUNK_LENGTH      32
/* 160 bit of SHA1 take 20 bytes */
#define SHA_DIGEST_SIZE          20

#define SRTP_POLICY_NO           0

setCryptoinfo::setCryptoinfo() {
}

setCryptoinfo::setCryptoinfo(const setCryptoinfo& orig) {
}

setCryptoinfo::~setCryptoinfo() {
}

```

```

}

uint8_t setCryptoinfo::getPolicyParamTypeValue(uint8_t policy_No, uint8_t
prot_type,
uint8_t policy_type){
    list<Policy_type *>::iterator i;
    for( i = policy.begin(); i != policy.end() ; i++ )
        if( (*i)->policy_No == policy_No && (*i)->prot_type == prot_type &&
(*i)->policy_type == policy_type && (*i)->length == 1)
            return (uint8_t)(*i)->value[0];

    switch(prot_type) {
    case MIKEY_PROTO_SRTP:
        if (policy_type < sizeof(srtpvalues)/sizeof(srtpvalues[0]))
            return srtpvalues[policy_type];
        printf("MIKEY_PROTO_SRTP type out of range %d", policy_type);
        break;
    case MIKEY_PROTO_IPSEC4:
        if (policy_type < sizeof(ipsec4values)/sizeof(ipsec4values[0]))
            return ipsec4values[policy_type];
        printf("MIKEY_PROTO_IPSEC4 type out of range %d", policy_type);
        break;
    default:
        break;
    }
    return 0;
}

/* Described in rfc3830.txt Section 4.1.2 */
void p( unsigned char * s, unsigned int sLength,
        unsigned char * label, unsigned int labelLength,
        unsigned int m,
        unsigned char * output )
{
    unsigned int i;
    unsigned int hmac_output_length;
    byte_t * hmac_input = new byte_t[ labelLength + SHA_DIGEST_SIZE ];

    /* initial step
    * calculate A_1 and store in hmac_input */

    hmac_sha1( s, sLength,
        label, labelLength,
        hmac_input, &hmac_output_length );
    assert( hmac_output_length == SHA_DIGEST_SIZE );
    memcpy( &hmac_input[SHA_DIGEST_SIZE], label, labelLength );

    /* calculate P(s,label,1)
    * and store in output[0 ... SHA_DIGEST_SIZE -1] */

    hmac_sha1( s, sLength,

```

```

    hmac_input, labelLength + SHA_DIGEST_SIZE,
    output, &hmac_output_length );
assert( hmac_output_length == SHA_DIGEST_SIZE );

/* need key-length > SHA_DIGEST_SIZE * 8 bits? */
for( i = 2; i <= m ; i++ )
{
    /* calculate A_i = HMAC (s, A_(i-1))
    * A_(i-1) is found in hmac_input
    * and A_i is stored in hmac_input,
    * important: label in upper indices [SHA_DIGEST_SIZE ... labelLength +
SHA_DIGEST_SIZE -1]
    * stays untouched and is repetitively reused! */

    hmac_sha1( s, sLength,
        hmac_input, SHA_DIGEST_SIZE,
        hmac_input, &hmac_output_length );
    assert( hmac_output_length == SHA_DIGEST_SIZE );

    /* calculate P(s,label,i), which is stored in
    * output[0 ... (i * SHA_DIGEST_SIZE) -1] */

    hmac_sha1( s, sLength,
        hmac_input, labelLength + SHA_DIGEST_SIZE,
        &output[ SHA_DIGEST_SIZE * (i-1) ], &hmac_output_length );
    assert( hmac_output_length == SHA_DIGEST_SIZE );
}

/* output now contains complete P(s,label,m)
* in output[0 ... (m * SHA_DIGEST_SIZE) -1] */
delete [] hmac_input;
}

/* Described in rfc3830.txt Section 4.1.2 */

void prf( unsigned char * inkey, unsigned int inkeyLength,
    unsigned char * label, unsigned int labelLength,
    unsigned char * outkey, unsigned int outkeyLength )
{
    unsigned int n, m, i, j;
    unsigned char * p_output;
    n = ( inkeyLength + PRF_KEY_CHUNK_LENGTH -1 )/
PRF_KEY_CHUNK_LENGTH;
    m = ( outkeyLength + SHA_DIGEST_SIZE -1 )/ SHA_DIGEST_SIZE;

    p_output = new unsigned char[ m * SHA_DIGEST_SIZE ];
    memset( outkey, 0, outkeyLength );
    for( i = 1; i <= n-1; i++ )
    {
        p( &inkey[ (i-1)*PRF_KEY_CHUNK_LENGTH ],
PRF_KEY_CHUNK_LENGTH, label, labelLength, m, p_output );

```

```

        for( j = 0; j < outkeyLength; j++ )
        {
            outkey[j] ^= p_output[j];
        }
    }

    /* Last step */
    p( &inkey[ (n-1)*PRF_KEY_CHUNK_LENGTH ], inkeyLength %
PRF_KEY_CHUNK_LENGTH, label, labelLength, m, p_output );

    for( j = 0; j < outkeyLength; j++ )
    {
        outkey[j] ^= p_output[j];
    }    delete [] p_output;
}

void setCryptoinfo::keyDeriv( unsigned char csId, unsigned int csIdValue,
unsigned char * inkey, unsigned int inkeyLength, uint8_t *randPtr, unsigned
int randLengthValue, unsigned char * key, unsigned int keyLength , int type ){

#ifdef SCSIM_SUPPORT
    if (dynamic_cast<SipSimSmartCardGD*>(*sim)){
        SipSimSmartCardGD *gd=dynamic_cast<SipSimSmartCardGD*>(*sim);
        gd->getKey(csId, csIdValue, (byte_t*)randPtr, randLengthValue, key,
keyLength, type);
    }else
#endif
    {
        byte_t * label = new byte_t[4+4+1+randLengthValue];
        switch( type ){
            case KEY_DERIV_SALT:
                label[0] = 0x39;
                label[1] = 0xA2;
                label[2] = 0xC1;
                label[3] = 0x4B;
                break;
            case KEY_DERIV_TEK:
                label[0] = 0x2A;
                label[1] = 0xD0;
                label[2] = 0x1C;
                label[3] = 0x64;
                break;
            case KEY_DERIV_TRANS_ENCR:
                label[0] = 0x15;
                label[1] = 0x05;
                label[2] = 0x33;
                label[3] = 0xE1;
                break;
            case KEY_DERIV_TRANS_SALT:
                label[0] = 0x29;
                label[1] = 0xB8;

```

```

        label[2] = 0x89;
        label[3] = 0x16;
        break;
    case KEY_DERIV_TRANS_AUTH:
        label[0] = 0x2D;
        label[1] = 0x22;
        label[2] = 0xAC;
        label[3] = 0x75;
        break;
    case KEY_DERIV_ENCR:
        label[0] = 0x15;
        label[1] = 0x79;
        label[2] = 0x8C;
        label[3] = 0xEF;
        break;
    case KEY_DERIV_AUTH:
        label[0] = 0x1B;
        label[1] = 0x5C;
        label[2] = 0x79;
        label[3] = 0x73;
        break;
    }

    label[4] = csId;

    label[5] = (unsigned char)((csbIdValue>>24) & 0xFF);
    label[6] = (unsigned char)((csbIdValue>>16) & 0xFF);
    label[7] = (unsigned char)((csbIdValue>>8) & 0xFF);
    label[8] = (unsigned char)(csbIdValue & 0xFF);
    memcpy( &label[9], randPtr, randLengthValue );
    prf( inkey, inkeyLength, label, 9 + randLengthValue, key, keyLength );
    delete [] label;
}
}

void setCryptoinfo::genTek( unsigned char csId, unsigned char * tek, unsigned int
tekLength, uint8_t *tgkPtr, uint8_t *rand, unsigned int csbIdValue ){
    keyDeriv( csId, csbIdValue, tgkPtr, 192, rand, 16, tek, tekLength,
KEY_DERIV_TEK );
}

void setCryptoinfo::genSalt( unsigned char csId, unsigned char * salt, unsigned int
saltLength, uint8_t *tgkPtr, uint8_t *rand, unsigned int csbIdValue ){
    keyDeriv( csId, csbIdValue, tgkPtr, 192, rand, 16, salt, saltLength,
KEY_DERIV_SALT );
}

void setCryptoinfo::genAuth( unsigned char csId, unsigned char * a_key, unsigned int
a_keylen, uint8_t *tgkPtr, uint8_t *rand, unsigned int csbIdValue ){
    keyDeriv( csId, csbIdValue, tgkPtr, 192, rand, 16, a_key, a_keylen,
KEY_DERIV_AUTH );
}

```

```

}

MRef<CryptoContext *> setCryptoInfo::initCrypto( uint32_t ssrc,
uint16_t seq_no, uint32_t f_roc, uint8_t *tgc, uint8_t *rand, unsigned int csbIdValue
){
    MRef<CryptoContext *> cryptoContext;
        cryptoContext = new CryptoContext( ssrc );

        unsigned char * masterKey = new unsigned char[16];
        unsigned char * authKey = new unsigned char[16];
        unsigned char * masterSalt = new unsigned char[14];
        uint8_t csId=1;
        uint32_t roc=f_roc;
        uint8_t policyNo = SRTP_POLICY_NO;
        //Extract Srtp policy !!! Check the return value if type not available
        uint8_t ealg = getPolicyParamTypeValue(policyNo,
MIKEY_PROTO_SRTP, MIKEY_SRTP_EALG);
        uint8_t ekeyl = getPolicyParamTypeValue(policyNo,
MIKEY_PROTO_SRTP, MIKEY_SRTP_EKEYL);
        uint8_t aalg = getPolicyParamTypeValue(policyNo,
MIKEY_PROTO_SRTP, MIKEY_SRTP_AALG);
        uint8_t akeyl = getPolicyParamTypeValue(policyNo,
MIKEY_PROTO_SRTP, MIKEY_SRTP_AKEYL);
        uint8_t skeyl = getPolicyParamTypeValue(policyNo,
MIKEY_PROTO_SRTP, MIKEY_SRTP_SALTKEYL);
        uint8_t keydr = getPolicyParamTypeValue(policyNo,
MIKEY_PROTO_SRTP, MIKEY_SRTP_KEY_DERRATE);
        uint8_t saltkeyl = getPolicyParamTypeValue(policyNo,
MIKEY_PROTO_SRTP, MIKEY_SRTP_SALTKEYL);
        uint8_t encr = getPolicyParamTypeValue(policyNo,
MIKEY_PROTO_SRTP, MIKEY_SRTP_ENCR_ON_OFF);
        uint8_t auth = getPolicyParamTypeValue(policyNo,
MIKEY_PROTO_SRTP, MIKEY_SRTP_AUTH_ON_OFF);
        uint8_t autht = getPolicyParamTypeValue(policyNo,
MIKEY_PROTO_SRTP, MIKEY_SRTP_AUTH_TAGL);

#ifdef ENABLE_TS
        ts.save("TEK_START");
#endif
        genTek( csId, masterKey, 16, tgc, rand, csbIdValue );
#ifdef ENABLE_TS
        ts.save("TEK_STOP");
#endif
        genSalt( csId, masterSalt, 14, tgc, rand, csbIdValue );

#ifdef DEBUG_OUTPUT
    #if 0
        fprintf( stderr, "csId: %i\n", csId );
        cerr << "SSRC: " << ssrc << " - TEK: " << binToHex( masterKey, 16 ) <<
endl;
        cerr << "SSRC: " << ssrc << " - SALT: " << binToHex( masterSalt, 14 ) <<
endl;
    #endif
}

```

```

#endif
#endif

master_salt_length, ekeyl, akeyl, skeyl, encr, auth, autht );
    cryptoContext = new CryptoContext( ssrc, roc, seq_no, keydr,
        ealg, aalg, masterKey, 16, masterSalt, 14, ekeyl, akeyl, skeyl, encr,
        auth, autht );

    cryptoContext->derive_srtp_keys( 0 );
    return cryptoContext;
}

void setCryptoinfo::handleRtpPacket( MRef<SRtpPacket *> packet,
uint8_t *tgk, uint8_t *rand, unsigned int csbIdValue){
    uint32_t packetSsrc; //this value needs to be generated from the packet
    uint16_t seq_no; //this value needs to be generated from the packet

    static int flag;
    static int roc;
    static int flag1=1;
    if( !packet ) {
        return;
    }
    packetSsrc = packet->getHeader().getSSRC();
    seq_no = packet->getHeader().getSeqNo();
    if(seq_no!=65535 && flag<flag1){
        roc=roc;
    }
    else if((seq_no==65535 )){
        roc=roc+1;
        flag=flag1;
        flag1++;
    }
    if( packet->unprotect(initCrypto(packetSsrc, seq_no, roc, tgk,rand, csbIdValue
))) {
        return;
    }
}

/*
* File: readescrowinfo.cpp
* Author: morshed
*
* Created on November 24, 2009, 10:44 PM
*/

#include </root/readwireshark/readEscrowinfo.h>
#include<fstream>
#include<iostream>
#include<stdio.h>
#include<stdlib.h>

using namespace std;

```



```

/*This class reads escrow info from the text File*/

readEscrowinfo::readEscrowinfo() {}
void readEscrowinfo::readEscrowinfo_fromFile() {

    FILE *fp;
    static int i, j, k, l; char c;
    int flag=1;
    fp=fopen("/home/morshed/Desktop/Lea/wiresharkfile.txt", "r");
    if(fp==NULL) {
        printf("file not foundyyy!\n");
        exit(0);
    }else {

        while(1){
            c=fgetc(fp);
            if(c!=EOF) {
                if (flag=1){
                    if (c!='%'){
                        tgk[i]=c;
                        i++;
                    }
                    else if(c=='%'){
                        flag=2;
                        i++;
                    }
                }
                else if(flag==2 ){
                    if(c!='%'){
                        rand[j]=c;
                        j++;
                    }
                    else if(c=='%'){flag=3;
                    j++;;}
                }
                else if(flag==3){
                    if(c!='%'){
                        csbidf[k]=c;
                        k++;
                    }
                    else if(c=='%'){flag=4;
                    }
                }
                else if(flag==4){

                    if(c!='%'){
                        signedhash[l]=c;
                        l++;
                    }
                    else if(c=='%'){flag=5;
                    }
                }
            }
        }
    }
}

```

```

        }
        else if (flag==5) break;
    }else break;
    }
    fclose(fp);
}
/*
 * File: mainclass.cpp
 * Author: morshed
 * Created on November 29, 2009, 10:40 PM
 */

#include <pcap.h>
#include<string.h>
#include "mainclass.h"
#include<libmutil/MemObject.h>
#include</usr/local/include/libminisip/media/rtp/RtpPacket.h>
#include<libminisip/media/rtp/SRtpPacket.h>
#include<libmcrypto/hmac.h>
#include<libminisip/media/rtp/CryptoContext.h>

#include</usr/include/boost/date_time.hpp>

using namespace std;
using namespace boost::posix_time;

mainclass::mainclass() {
}

mainclass::mainclass(const mainclass& orig) {
}

mainclass::~~mainclass() {
}

void read_packet (u_char *useless, const struct pcap_pkthdr *phdr, const u_char
*pkt){

    //tgk, rand, csbid, csid will be read here and will be pass through process packet
    //processpacket function further will send these value to handle packet for
//processing
    ptime time_start(microsec_clock::local_time());
    u_char *buf;
    int i=0,j;
    int p_len=phdr->len;

    /* in buf only encrypted payloads entered*/

    buf=new u_char[p_len-43];
    for (i=44, j=0;i<p_len;i++,j++)

```

```

        buf[j]=pkt[i];
    MRef <SRtpPacket *> packet;
    packet=SRtpPacket::readPacketF1(buf,j);
    readEscrowInfo escrow;
    // following line can be used if a static TGK value is used
    /*unsigned char tkg_based64[]="LibgYDfqyX8810mbqUjMYPXVJMu4dnT/sa8q
NDk8YqjSwC5t0MMMgVFMGDuQlbBpxd7z01ka2bkPJw49G4kX34VlGhwIJ
gIOU4rS1H1W9MPWIKLT+2p3vHAnK9uEXZPiqO/Cz5pvXnTRY/ySG5o1+
Yx4m6CDePqd7IgwY5id+/f92NZNoF3kd+v/9IX4gIVebM4f6OpI6wr9XpVs5K
G196e1g/eIQCGuancnO36y59i/TQww5W3FnTUUyRZCDTd";*/
    unsigned char *tkg_based64=new unsigned char[256];
    escrow.tkg_based64;
    int *tgklen=new int[192];
    unsigned char *tgk= new unsigned char[192];
        tkg= base64_decode(tkg_based64, 256, tgklen);
    // unsigned char rand_based64[]="S0iA5gDCwhfcseEdvO5rXg==";
    unsigned char *rand_based64=new unsigned char[24];
    escrow.rand_based64;
    int randlen=16;
    unsigned char *rand= new unsigned char[randlen];
    rand= base64_decode(rand_based64, 24, &randlen);
    unsigned int csbid=5862418;
    unsigned char * hmacAuthKey;
    setCryptoInfo sc;
    sc.handleRtpPacket(packet, tkg, rand, csbid);
    ptime time_end(microsec_clock::local_time());
    time_duration duration(time_end-time_start);
    cout<<duration<<"\n";
    printf("\n");
}
void mainclass::run()
{
    pcap_t *pcap;
    char pcapErr[PCAP_ERRBUF_SIZE];
    char fname[]="/home/morshed/Desktop/Lea/wireshark/srtp1.libpcap";
    pcap= pcap_open_offline(fname, pcapErr);
    if (pcap == NULL)
    {
        fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
        exit(EXIT_FAILURE);
    }
    ptime time_start(microsec_clock::local_time());
    pcap_loop(pcap, 0, read_packet, NULL);
    ptime time_end(microsec_clock::local_time());
    time_duration duration(time_end-time_start);
    cout<<"time:: " <<duration;
    pcap_close(pcap);
}

```

C. Source code of Verification Module

```
/*
 * File: setsig.cpp
 * Author: morshed
 *
 * Created on December 22, 2009, 1:28 AM
 */

#include "setsig.h"

using namespace std;

setsig::setsig() {
}
/*This function inserts signature from the RTCP packet to a vector*/

setsig::setsig(unsigned char *sigvalue, unsigned int siglength) {
    this->sigv = new unsigned char[ siglength ];
    memcpy( this->sigv, sigvalue, siglength);
}
setsig::setsig(const setsig& orig) {
}

setsig::~setsig() {
}
/*
 * File: sethash.cpp
 * Author:morshed
 *
 * Created on December 21, 2009, 12:58 PM
 */

#include "sethash.h"

using namespace std;

/*This function insert hash of SRTP block into a vector*/

sethash::sethash() {
}
sethash::sethash(unsigned char *hashvalue, unsigned int hashlength) {
    this->hashv = new unsigned char[ hashlength ];
    memcpy( this->hashv, hashvalue, hashlength);
}

sethash::sethash(const sethash& orig) {
```

```

}

sethash::~sethash() {
}
/*
 * File: VerificationRun.cpp
 * Author: morshed
 *
 * Created on December 13, 2009, 10:16 PM
 */

#include "VerificationRun.h"
#include "sethash.h"
#include<iostream>
#include<stdlib.h>
#include<string.h>
#include <pcap.h>
#include<libmutil/MemObject.h>
#include</usr/local/include/libminisip/media/rtp/RtpPacket.h>
#include<libminisip/media/rtp/SRtpPacket.h>
#include<libmcrypto/hmac.h>
#include<libminisip/media/rtp/CryptoContext.h>

uint16_t sequence_no=29115, f_seq;
static int count,q, blockSize, c=1,j,block_number, f_no;
unsigned char * rawhashdata, *hashValue;
unsigned int hashLength;

std::list< MRef<sethash *>> hash;
std::list< MRef<setsig *>> sigdata;
int er,err, forged_block[100],fogery_info[200];

using namespace std;

VerificationRun::VerificationRun() {

}

VerificationRun::VerificationRun(const VerificationRun& orig) {
}

VerificationRun::~~VerificationRun() {
}

unsigned char * VerificationRun::CreateHash(){
    hashValue = new unsigned char [20];
    readEscrow escrow;
    //following line can be used for static value of TGK

```

```

/* unsigned char
tgk_based64[]="22XcP0y/iJ9LpDzj0NVT+JDXZrxDwRurmuX6sWqPihp
K5jMaGXtHUhu3hFKVj3EfCIGBSExIy9ZY0Uunn8ZjzFVtJHkdohGXNE+q8e/JR
zdexc6mFKCO0F9V7y3RfnPiTQHcSpqbX6v9aaQEGbYTe/BVn328QUgWruYp80v
4YPCARRngAOJFQ4wHHIS62dXPB79XVckBbCMntLYeOdr9q21HSBDW0ljJyG
L49FBM+bukuPlaKKGViaEhpJtZprR";*/
tgk_based64[]=escrow.tgkf;
int *tgklen=new int[192];
unsigned char *tgk= new unsigned char[192];
tgk= base64_decode(tgk_based64, 256, tgklen);
//following line can be used for static value of rand value
/* unsigned char rand_based64[]="LVLDXNAEr2xiDcpzIKk0Yw==";*/
unsigned char rand_based64[]=escrow.randf;
int randlen=16;
unsigned char *rand= new unsigned char[randlen];
rand= base64_decode(rand_based64, 24, &randlen);
//following line can be used for static value of csbid
/* unsigned int csbid=-1038922046; */
unsigned int csbid=escrow.csbidf;
unsigned char * hmacAuthKey;
unsigned char csId=1;
setCryptoinfo sc;
    hmacAuthKey = new unsigned char[AUTH_KEY_SIZE];
    sc.genAuth(csId, hmacAuthKey, AUTH_KEY_SIZE,tgk,rand,csbid);
    for(int i=0;i<AUTH_KEY_SIZE;i++)
hmac_sha1( hmacAuthKey, AUTH_KEY_SIZE,
            (unsigned char *)rawhashdata, /*data*/
            blockSize, /*authenticated part length*/
            hashValue, /*tag*/
            &hashLength);
}

void VerificationRun::ProcessBlock(MRef<SRtpPacket *> pkt){

MRef<sethash*> hs;
if (count == 0)
    rawhashdata = new unsigned char [BLOCK_SIZE*pkt->size()];
    memcpy( &rawhashdata[blockSize], pkt->getBytes(), pkt->size());

    blockSize += pkt->size();
    count++;j++;

    if (count >= BLOCK_SIZE) {
        CreateHash();
        hs=new sethash(hashValue, hashLength);
        hash.push_back(hs);
        count=0;
    }
}

```

```

        blockSize=0;

        delete [] rawhashdata;
        delete [] hashValue;
    }
}

void VerificationRun::setvec(unsigned char *sigd, unsigned int len){

    MRef<setsig *> rp;

    rp=new setsig(sigd, len);
    massert(rp);

    if(!rp.isNull())
        sigdata.push_back(rp);
    else
        cout<<"cannot create signature vector"<<endl;
}

void VerificationRun::showsigdata (){

    list<MRef<setsig *>> ::iterator i;
    unsigned char *sig_vector=new unsigned char[128];
    if(!sigdata.empty()) {
        for( i = sigdata.begin(); i!= sigdata.end(); i++){

            sig_vector=(*i)->getsig();
            for(int j=0;j<128;j++)
                printf(" %x",sig_vector[j]);
        }
    }
    else cout<<"signature hash empty"<<endl;
}

void VerificationRun::verifyHash(){

    list<MRef<sethash *>> ::iterator i;
    unsigned char *hash_vector=new unsigned char[23296];
    for( i = hash.begin(); i!= hash.end(); i++){

        hash_vector=(*i)->gethash();
        for(int j=0;j<23296;j++)

```

```

        printf(" %x",hash_vector[j]);
    }
}

void store_block (u_char *useless, const struct pcap_pkthdr *phdr, const u_char
*pkt){

    int i=0,j;
    int p_len=phdr->len;

    /* in buf only encrypted payloads entered*/
    u_char *buf=new u_char[p_len-43];
    for (i=44, j=0;i<p_len;i++,j++)

        buf[j]=pkt[i];

    MRef <SRtpPacket *> packet;
    packet=SRtpPacket::readPacketF1(buf,j);
    VerificationRun v;
    v.ProcessBlock(packet);
    delete [] buf;
}

//if first sequence number is escrowed then the following function will work

//void firstsequence_callback (u_char *useless, const struct pcap_pkthdr *phdr, const
u_char *pkt){
//
//  int i=0,j;
//  int p_len=phdr->len;
//
//  /* in buf only encrypted payloads entered*/
//  u_char *buf=new u_char[p_len-43];
//  for (i=44, j=0;i<p_len;i++,j++)
//
//      buf[j]=pkt[i];
//
//  MRef <SRtpPacket *> packet;
//  // cout<<"ok"<<endl;
//  packet=SRtpPacket::readPacketF1(buf,j);
//  VerificationRun v;
//  if(packet->getHeader().getSeqNo()==sequence_no){
//      v.ProcessBlock(packet);
//      sequence_no++;
//      f_seq=sequence_no;
//  }
//  else if(packet->getHeader().getSeqNo()==sequence_no+1){
//      cout<<"invalid sequence no.,packet missing"<<endl;

```



```

//     sequence_no++;
//     count++;
//     // blockSize += packet->size();
// }
// else if(packet->getHeader().getSeqNo()!=sequence_no){
//     cout <<"invalid sequence no, forged packet may be inserted in the captured file
at block no:: "<<block_number<<endl;
////     v.ProcessBlock(packet);
//     count++;
//     sequence_no=sequence_no+1;
// }
// delete [] buf;
//}

void rtcvector(u_char *useless, const struct pcap_pkthdr *phdr, const u_char *pkt){
    MRef<setsig*> rs;
    int i=0,j,p;
    int p_len=phdr->len;
    unsigned char *bufRtcp;

    /* in buf only encrypted payloads entered*/

    bufRtcp=new u_char[p_len-44];
    for (i=44, j=0; i<p_len-4; i++, j++)
        bufRtcp[j]=(unsigned char)pkt[i];

    rs=new setsig(bufRtcp, j);
    sigdata.push_back(rs);
    delete [] bufRtcp;
}

//void pRtcp_callback (u_char *useless, const struct pcap_pkthdr *phdr, const u_char
*pkt){
void verify_hash(){
    int i=0,j,p;

    list<MRef<sethash *>> ::iterator k;
    list<MRef<setsig *>> ::iterator m;

    unsigned char *hash_vector=new unsigned char[20];

    for( k = hash.begin(),p=0; k!= hash.end(); k++,p++){
        hash_vector=(*k)->gethash();
        for( m = sigdata.begin(),j=0; m!= sigdata.end(); m++,j++){

            MRef<Certificate*> cert;

```

```

cert=OsslCertificate::load("/home/morshed/trunk/minisip/test_cert/alice_cert.pem");
//cert= new
OsslCertificate("/home/morshed/trunk/minisip/test_cert/alice_cert.pem");

    massert(cert);

// er=cert->verifSign(hash_vector, 20, bufRtcp, 128);
er=cert->verifSign(hash_vector, 20, (*m)->getsig(), 128);

    if(er==1 && m!= sigdata.end()){
//      cout<<endl<<"No forgery made in block no. :: "<<p<<endl;
      block_number++;
      err=1;
//      forgery_info[f_no]=3;
//      f_no++;
      printf("3\n");
      delete []hash_vector;
      break;
    }

}
if(er==0){
//  forgery_info[f_no]=3;
//  f_no++;
  printf("5\n");
//  cout<<endl<<"Forgery made in block no. :: "<<p<<endl;
}

}
}

void VerificationRun::run(){

    pcap_t *pcap;
    char pcapErr[PCAP_ERRBUF_SIZE];
//char fname[]="/home/morshed/Desktop/Lea/readsrtp.libpcap";
//char fname[]="/home/morshed/Desktop/Lea/wireshark/srtp1.libpcap";
//char fname[]="/home/morshed/Desktop/Lea/srtp1.libpcap";
//char fname[]="/home/morshed/Desktop/Lea/wireshark/srtp8.libpcap";
    char fname[]="/home/morshed/Desktop/Lea/wireshark/srtp_forged1.libpcap";

    pcap= pcap_open_offline(fname, pcapErr);

    if (pcap == NULL)
    {
        fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
        exit(EXIT_FAILURE);
    }
}

```

```

pcap_loop(pcap, 0, store_block, NULL);
pcap_close(pcap);
// following three lines deals with the last block if the last block<blocksize
CreateHash();
// new sethash(hashValue, hashLength);
hash.push_back(new sethash(hashValue, hashLength));

delete [] rawhashdata;
delete [] hashValue;

}

void VerificationRun::runRtcp(){

pcap_t *pcap_rtcp;
char pcapErr[PCAP_ERRBUF_SIZE];

char frtcp[]="/home/morshed/Desktop/Lea/wireshark/rtcpf1.libpcap";
//char frtcp[]="/home/morshed/Desktop/Lea/readrtcp.libpcap";

pcap_rtcp= pcap_open_offline(frtcp, pcapErr);

VerificationRun r;
r.run();
if (pcap_rtcp == NULL)
{
    fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
    exit(EXIT_FAILURE);
}
//pcap_loop(pcap_rtcp, 0, pRtcp_callback, NULL);
pcap_loop(pcap_rtcp, 0, rtcpvector, NULL);
verify_hash();

pcap_close(pcap_rtcp);
}

```

D. Source code of Attacker Module

```

/* File: setVecotr.cpp*/

#include "setVecotr.h"
#include<libmutil/MemObject.h>
#include<string.h>

using namespace std;
setVecotr::setVecotr() {
}

/*This function inserts hash of a block into the vector*/

setVecotr::setVecotr(unsigned char *blockvalue, unsigned int blocklength) {
    this->blockv = new unsigned char[ blocklength ];
    memcpy( this->blockv, blockvalue, blocklength);
}

setVecotr::setVecotr(const setVecotr& orig) {
}

setVecotr::~setVecotr() {
}

/* File: mainattacker.cpp*/

#include "mainattacker.h"
#include</root/libattacker/setVecotr.h>

using namespace std;

static int place, count=0;
char frtcp[]="/home/morshed/Desktop/Lea/wireshark/srtp_forged5.libpcap";
pcap_dumper_t *pd, *pd1;
std::list< MRef<setVecotr *>> block;

mainattacker::mainattacker() {
}

mainattacker::mainattacker(const mainattacker& orig) {
}

mainattacker::~mainattacker() {
}

/*This function dumps packet or block without SRTP authentication and called by

```

```

insert_block_packet() function*/
void pdump (u_char *pd, const struct pcap_pkthdr *phdr, const u_char *pkt){
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}

/*
* This function insert a packet or block at the front, middle and last. It is needs
* to make a libpcap file (by which actual captured file will be forged) consisting of
* one packet to insert on packet or to make a libpcap file consisting of one block to
* insert a full block in the captured file (this is without SRTP authentication).
*/

void insert_block_packet (u_char *pd, const struct pcap_pkthdr *phdr, const u_char
*pkt){

    if (count==place & count<=128){
        pcap_t *pcap_filefromwright1;
        char pcapErr[PCAP_ERRBUF_SIZE];
        char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1b.libpcap"; /* for
block, srtp1b contains block*/
/*      char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1p.libpcap"; *//*for
packet, srtp1p contains a single packet */

        pcap_filefromwright1= pcap_open_offline(frtcp2, pcapErr);

        if (pcap_filefromwright1 == NULL)
        {
            fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
            exit(EXIT_FAILURE);
        }
        if(count==0){
            pcap_loop(pcap_filefromwright1, 0, pdump, (u_char *)pd);//(1*) this line add
ablock or packet not replace
            pcap_dump((u_char*)pd, phdr, pkt );
            count++;
        }
        else{
            //pcap_dump((u_char*)pd, phdr, pkt );
            pcap_loop(pcap_filefromwright1, 0, pdump, (u_char *)pd);
            count++;
        }
    }
    else {
        pcap_dump((u_char*)pd, phdr, pkt );
        count++;
    }
}

```

```

}

/*This function dumps packet or block with SRTP authentication and called by
insert_block_packet_srtp() function*/
void pdump_srtp (u_char *pd, const struct pcap_pkthdr *phdr, const u_char *pkt){

    u_char *buf=(u_char*)pkt;
    MRef<SRtpPacket *> packet;
    packet=SRtpPacket::readPacketF1(buf,phdr->len);
    buf=packet->getContent();
    pcap_dump((u_char*)pd, phdr, pkt);
    //place++;
    //count++;
}

/*
* This function insert a packet or block at the front, middle and last. It is needs
* to make a libpcap file (by which actual captured file will be forged) consisting of
* one packet to insert on packet or to make a libpcap file consisting of one block to
* insert a full block in the captured file (this is with SRTP authentication).
*/

void insert_block_packet_Srtp (u_char *pd, const struct pcap_pkthdr *phdr, const
u_char *pkt){

    if (count==place){
        pcap_t *pcap_filefromwright1;
        char pcapErr[PCAP_ERRBUF_SIZE];

// for block, srtp1b contains block
//   char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1b.libpcap";

//for packet, srtp1p contains a single packet
        char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1p.libpcap";
        pcap_filefromwright1= pcap_open_offline(frtcp2, pcapErr);

        if (pcap_filefromwright1 == NULL)
        {
            fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
            exit(EXIT_FAILURE);
        }
        if(count==0){
            // this line add a block or packet not replace
            pcap_loop(pcap_filefromwright1, 0, pdump_srtp, (u_char *)pd);
            pcap_dump((u_char*)pd, phdr, pkt );
            count++;
        }
    }
}

```

```

else{
    pcap_dump((u_char*)pd, phdr, pkt );
    pcap_loop(pcap_filefromwright1, 0, pdump_srtp, (u_char *)pd);
    //pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}
}
else {
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}
}

/* following function replace the content of a packet along with sequence no. and
* without SRTP authentication.
*/
void pd_callback (u_char *pd, const struct pcap_pkthdr *phdr, const u_char *pkt){

    int i=0,j,p;
    int p_len=phdr->len;

    /* in buf only encrypted payloads entered*/
    u_char *buf=(u_char*)pkt;

    for (i=44;i<p_len-4;i++)
        buf[i]='d';

    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}

void po_callback (u_char *pd, const struct pcap_pkthdr *phdr, const u_char *pkt){

//    pcap_t *pcap_filefromwright1;
//    char pcapErr[PCAP_ERRBUF_SIZE];

    pcap_dump((u_char*)pd, phdr, pkt );
    count++;

    //char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1b.libpcap";
    //char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1p.libpcap";
    //pcap_filefromwright1= pcap_open_offline(frtcp2, pcapErr);
    //
    //if (pcap_filefromwright1 == NULL)
    //{
    //    fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
    //    exit(EXIT_FAILURE);
}

```

```

//    }
//    pcap_loop(pcap_filefromwright1, 0, pcap_dump, (u_char *)pd);
//    printf("packet inserted successfully\n");
////    pcap_breakloop(pcap_filefromwright);
////    exit(0);
//    count++;
//
//    pcap_dump((u_char*)pd, phdr, pkt );
//    count++;
//
////
}

//this following module for replacing a block
/*
void pblock_callback (u_char *pd, const struct pcap_pkthdr *phdr, const u_char
*pkt){

    if (count==place){
        pcap_t *pcap_filefromwright1;
        char pcapErr[PCAP_ERRBUF_SIZE];

        pcap_dump((u_char*)pd, phdr, pkt );

        //char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1b.libpcap";
        char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1b.libpcap";
        pcap_filefromwright1= pcap_open_offline(frtcp2, pcapErr);

        if (pcap_filefromwright1 == NULL)
        {
            fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
            exit(EXIT_FAILURE);
        }
        pcap_loop(pcap_filefromwright1, 0, po_callback, (u_char *)pd);
        printf("packet inserted successfully\n");
    }
    else{
        pcap_dump((u_char*)pd, phdr, pkt );
        count++;
    }
}
*/

//this following pv_callback function replace a packet
//void pv_callback (u_char *pd, const struct pcap_pkthdr *phdr, const u_char *pkt){
//
//    if (count==place){
//        pcap_t *pcap_filefromwright1;
//        char pcapErr[PCAP_ERRBUF_SIZE];

```



```

//
// //pcap_dump((u_char*)pd, phdr, pkt );//(1*)this line used for inserting a packet
or block, not for replacing a packet or block
//
// //char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1b.libpcap";
// char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1p.libpcap";
// pcap_filefromwright1= pcap_open_offline(frtcp2, pcapErr);
//
// if (pcap_filefromwright1 == NULL)
// {
//     fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
//     exit(EXIT_FAILURE);
// }
// //pcap_loop(pcap_filefromwright1, 0, pcap_dump, (u_char *)pd);//(1*) this line
add ablock or packet not replace
// // pcap_loop(pcap_filefromwright1, 0, pcap_dump, (u_char *)pd);//this line
replace a packet
//     pcap_loop(pcap_filefromwright1, 0, pdump, (u_char *)pd);
//     printf("packet inserted successfully\n");
////     pcap_breakloop(pcap_filefromwright);
////     exit(0);
//     count++;
// }
// else{
//     pcap_dump((u_char*)pd, phdr, pkt );
//     count++;
// }
//// printf("successful");
//}

void pv_callback (u_char *pd, const struct pcap_pkthdr *phdr, const u_char *pkt){

    if (count==place){
        pcap_t *pcap_filefromwright1;
        char pcapErr[PCAP_ERRBUF_SIZE];

        //pcap_dump((u_char*)pd, phdr, pkt );//(1*)this line used for inserting a packet
or block, not for replacing a packet or block

        char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1b.libpcap";
// char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1p.libpcap";
        pcap_filefromwright1= pcap_open_offline(frtcp2, pcapErr);

        if (pcap_filefromwright1 == NULL)
        {
            fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
            exit(EXIT_FAILURE);
        }
    }
}

```

```

    }
    //pcap_loop(pcap_filefromwright1, 0, pcap_dump, (u_char *)pd);//(1*) this line
add ablock or packet not replace
    // pcap_loop(pcap_filefromwright1, 0, pcap_dump, (u_char *)pd);//this line
replace a packet
    // pd1=pd[place];

//this block is for replacing a block
    pcap_loop(pcap_filefromwright1, 0, po_callback, (u_char *)pd);
//this block is for replacing the content of a packet
    //pcap_loop(pcap_filefromwright1, 0, pc_callback, (u_char *)pd);
    printf("packet inserted successfully\n");
}
else if(count>=place+128){
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}else count++;
}

//this function for replacing content of a packet
void pc_callback (u_char *pd, const struct pcap_pkthdr *phdr, const u_char *pkt){
    if (count==place){
        pcap_t *pcap_filefromwright1;
        char pcapErr[PCAP_ERRBUF_SIZE];

        char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1p.libpcap";
        pcap_filefromwright1= pcap_open_offline(frtcp2, pcapErr);

        if (pcap_filefromwright1 == NULL)
        {
            fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
            exit(EXIT_FAILURE);
        }
        //pcap_loop(pcap_filefromwright1, 0, pcap_dump, (u_char *)pd);//(1*) this line
add ablock or packet not replace
        // pcap_loop(pcap_filefromwright1, 0, pcap_dump, (u_char *)pd);//this line
replace a packet
        // pd1=pd[place];
        pcap_loop(pcap_filefromwright1, 0, pd_callback, (u_char *)pd); //this block is
for replacing a block
        //pcap_loop(pcap_filefromwright1, 0, pc_callback, (u_char *)pd); //this block is
for replacing the content of a packet
        printf("packet inserted successfully\n");
    }
    else {
        pcap_dump((u_char*)pd, phdr, pkt );
        count++;
    }
}

```

```

}

//This function replace the contend of the funnction without replacing sequece no.'
// and call the ps_callback function and also without SRTP authentication data

void pcps_callback (u_char *pd, const struct pcap_pkthdr *phdr, const u_char *pkt){

    if (count==place){

        int p_len=phdr->len;

        /* in buf only encrypted payloads entered*/
        u_char *buf=(u_char*)pkt;

        // for (int i=48;i<p_len-4;i++)//only sequence no. is not modified
        for (int i=56;i<p_len-4;i++) //only data is modified
            buf[i]='d';

        pcap_dump((u_char*)pd, phdr, pkt );
        count++;
    }
    else {
        pcap_dump((u_char*)pd, phdr, pkt );
        count++;
    }
}

//This function replace the contend of the funnction without replacing sequece no.'
// and call the ps_callback function and also with SRTP authentication data

void pcas_callback (u_char *pd, const struct pcap_pkthdr *phdr, const u_char *pkt){

    if (count==place){

        int p_len=phdr->len;

        /* in buf only encrypted payloads entered*/
        u_char *buf=(u_char*)pkt;

        // for (int i=48;i<p_len-4;i++)//only sequence no. is not modified
        for (int i=56;i<p_len-4;i++) //only data is modified
            buf[i]='d';
        MRef <SRtpPacket *> packet;

        packet=SRtpPacket::readPacketF1(buf,182);
        buf=packet->getContent();
        pcap_dump((u_char*)pd, phdr, pkt );
        count++;
    }
}

```

```

    }
    else {
        pcap_dump((u_char*)pd, phdr, pkt );
        count++;
    }
}

//This function replace the contend of the funnction replacing seqence no.'
// and call the ps_callback function and also with SRTP authentication data

void psrtp_auth_callback (u_char *pd, const struct pcap_pkthdr *phdr, const u_char
*pkt){

    if (count==place){

        int p_len=phdr->len;

        /* in buf only encrypted payloads entered*/
        u_char *buf=(u_char*)pkt;

        // for (int i=48;i<p_len-4;i++)//only sequence no. is not modified
        for (int i=46;i<p_len-4;i++) //only data is modified
            buf[i]='d';
        MRef <SRtpPacket *> packet;

        packet=SRtpPacket::readPacketF1(buf,182);
        buf=packet->getContent();
        pcap_dump((u_char*)pd, phdr, pkt );
        count++;
    }
    else {
        pcap_dump((u_char*)pd, phdr, pkt );
        count++;
    }
}

//replacing block function2. This creates the vector of the packet
void rblock_callback (u_char *p, const struct pcap_pkthdr *p_hdr, const u_char
*pkbtb){

    u_char *buf=(u_char *)pkbtb;
    for(int i=0,j=0;i<101;i++,j++)
        buf[j]=pkbtb[i];
    MRef<setVecotr*> replaced_p;
    replaced_p=new setVecotr(buf, p_hdr->len);
    block.push_back(replaced_p);
}

```

```

//replacing a block function1
void rblock (){

    pcap_t *pcap_filefromwright1;
    char pcapErr[PCAP_ERRBUF_SIZE];

    char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp4.libpcap";
    pcap_filefromwright1= pcap_open_offline(frtcp2, pcapErr);

    if (pcap_filefromwright1 == NULL)
    {
        fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
        exit(EXIT_FAILURE);
    }
    pcap_loop(pcap_filefromwright1, 0, rblock_callback, NULL);
    pcap_close(pcap_filefromwright1);
}

/*This function calculates the UDP checksum*/

unsigned short computeUDPChecksum(unsigned short udp_len, unsigned short
src_add[],unsigned short dest_add[],
    unsigned short no_of_dataoctet, unsigned short data[])
{
    unsigned short udp_port_no=17;
    unsigned short padd=0;
    unsigned short adjacent_two_octet;
    unsigned long sum,checksum;
    bool padding;

    //determine padding
    if (((no_of_dataoctet-1) % 2)!=0)
        padding=1;
    else padding=0;

    // Find out if the length of data is even or odd number. If odd,
    // add a padding byte = 0 at the end of packet
    if (padding&1==1){
        padd=1;
        data[udp_len]=0;
    }

    //initialize sum to zero
    sum=0;

```

```

// make 16 bit words out of every two adjacent 8 bit words and
// calculate the sum of all 16 vit words
for (int i=0;i<udp_len+padd;i=i+2){
    adjacent_two_octet =((data[i]<<8)&0xFF00)+(data[i+1]&0xFF);

    sum = sum + (unsigned long)adjacent_two_octet;
}
//_ add the UDP pseudo header which contains the IP source and destinationn
addresses
for (int i=0;i<4;i=i+2){
    adjacent_two_octet =((src_add[i]<<8)&0xFF00)+(src_add[i+1]&0xFF);
    sum=sum+adjacent_two_octet;
}
for (int i=0;i<4;i=i+2){
    adjacent_two_octet =((dest_add[i]<<8)&0xFF00)+(dest_add[i+1]&0xFF);
    sum=sum+adjacent_two_octet;
}
// the protocol number and the length of the UDP packet
sum = sum + udp_port_no+ udp_len;

// keep only the last 16 bits of the 32 bit calculated sum and add the carries
while (sum>>16)
    sum = (sum & 0xFFFF)+(sum >> 16);

// Take the one's complement of sum
checksum = ~sum;

return ((unsigned short) checksum);
}

//This function replace the contend of a paccket without replacing seqence no.'
// and call the ps_callback function and also without SRTP authentication data
void replace_data (u_char *pd, const struct pcap_pkthdr *phdr, const u_char *pkt){

    unsigned short *src_add=new unsigned short[8];
    unsigned short *dst_add=new unsigned short[8];
    unsigned short *data=new unsigned short[226];
    unsigned short checksum;
    if (count==place){

        int t, p_len=phdr->len;

        /* in buf only encrypted payloads entered*/
        u_char *buf=(u_char*)pkt;
        buf[42]=0;
        buf[43]=0;

```

```

// for (int i=48;i<p_len-4;i++)//only sequence no. is not modified
for (int i=56;i<p_len;i++){ //only data is modified
    buf[i]='a';
}
for(int i=36,j=0;i<p_len;i++,j++){
    data[j]=pkt[i];
    t++;
}
for(int i=28,j=0;i<32;i++,j++){
    src_add[j]=pkt[i]; //convert unsigned char * to unsigned short
}
for(int i=32,j=0;i<36;i++,j++){
    dst_add[j]=pkt[i];
}
//unsigned short *p = static_cast<unsigned
short*>(static_cast<void*>(&packetBuffer[1]));
//checksum=udp_sum_calc(190, src_add, dst_add, 1, data);
checksum=computeUDPChecksum(190, src_add, dst_add, t, data);
printf("%d\n",checksum);
u_char *chksum=new u_char[4];

//converts checksum fro unsigned short to unsigned char*

chksum[0] = (unsigned char)((checksum>>8) & 0xFF);
chksum[1] = (unsigned char)(checksum & 0xFF);

buf[42]=chksum[0];
buf[43]=chksum[1];
pcap_dump((u_char*)pd, phdr, pkt );
count++;
}
else {
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}
}

/*This function replace the content of a packet without sequence no as well as SRTP
authentication
* this function can also be used to replace the content of a block of packets changing
the input file.
*/

void replace_content_packet (u_char *pd, const struct pcap_pkthdr *phdr, const
u_char *pkt){

    if (count==place){

```

```

pcap_t *pcap_filefromwright1;
char pcapErr[PCAP_ERRBUF_SIZE];

char frtcp2[]="/home/morshed/Desktop/Lea/wireshark/srtp1p.libpcap";
pcap_filefromwright1= pcap_open_offline(frtcp2, pcapErr);

if (pcap_filefromwright1 == NULL)
{
    fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
    exit(EXIT_FAILURE);
}

pcap_loop(pcap_filefromwright1, 0, pd_callback, (u_char *)pd);

}
else {
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}
}

/*ok function
void replace_block_withoutsrtpauth (u_char *pd, const struct pcap_pkthdr *phdr,
const u_char *pkt){
    int p,t=0,i,j; u16 *data=new u16[101];
    MRef <SRtpPacket *> packet;

    u16 *src_add=new u16[8];
    u16 *dst_add=new u16[8];
    u16 checksum;
    if (count==place & place<=10){
        list<MRef<setVecotr *>> ::iterator k;
        // u_char *buf=new u_char[phdr->len];
        u_char * dt=(u_char *)pkt;

        dt[40]=0;
        dt[41]=0;
        for( i=34,j=0;i<phdr->len;i++,j++){
            data[j]=((dt[i]));
            t++;
        }
        for(int i=26,j=0;i<30;i=i++,j++){
            src_add[j]=((dt[i])); //convert unsigned char * to unsigned short
            printf("%x ",src_add[j]);
        }

        for(int i=30,j=0;i<34;i=i++,j++){

```



```

        dst_add[j]=((dt[i]));
    }
    //unsigned short *p = static_cast<unsigned
short*>(static_cast<void*>(&packetBuffer[1]));

    checksum=udp_sum_calc(67, src_add, dst_add, 0, data);

    // u_char *chksum=static_cast<u16>(static_cast<void>(&checksum));
    u_char *chksum=new u_char[4];

    //converts checksum fro unsigned short to unsigned char*

    chksum[0] = (unsigned char)((checksum>>8));
    chksum[1] = (unsigned char)checksum;

    dt[40]=chksum[0];
    dt[41]=chksum[1];
    //for new checksum

    pcap_dump((u_char*)pd, phdr, pkt );
    place++;
    count++;
}
else {
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}
}
*/

//replacing block function3 with srtp authentication with checksum
void replace_block_withsrtpauth (u_char *pd, const struct pcap_pkthdr *phdr, const
u_char *pkt){
    int p, t=0; u16 *data=new u16[226];
    MRef <SRtpPacket *> packet;

    u16 *src_add=new u16[8];
    u16 *dst_add=new u16[8];
    u16 checksum;
    if (count==place & place<=128){
        list<MRef<setVecotr *>> ::iterator k;
        u_char * buf=(u_char *)pkt;

        unsigned char *bl_vector=new unsigned char[phdr->len];

        for( k = block.begin(),p=1; k!= block.end(); k++,p++ ){
            if (place==p){

```

```

        bl_vector=(*k)->getblock();
        // for(int i=0;i<phdr->len;i++) //if full packet is replaced
        for(int i=44;i<phdr->len;i++)
            buf[i]=bl_vector[i];
    }
}
buf[42]=0;
buf[43]=0;
packet=SRtpPacket::readPacketF1(buf,phdr->len);
char *dt=new char[packet->size()];
dt=packet->getBytes();
for(int i=36,j=0;i<packet->size();i++,j++){
    data[j]=dt[i];
    t++;
}

for(int i=28,j=0;i<32;i++,j++){
    src_add[j]=dt[i]; //convert unsigned char * to unsigned short
}
for(int i=32,j=0;i<36;i++,j++){
    dst_add[j]=dt[i];
}
//unsigned short *p = static_cast<unsigned
short*>(static_cast<void*>(&packetBuffer[1]));

checksum=computeUDPChecksum(190, src_add, dst_add, t, data);

// u_char *chksum=static_cast<u16>(static_cast<void>(&checksum));
u_char *chksum=new u_char[4];

//converts checksum fro unsigned short to unsigned char*

    chksum[0] = (unsigned char)((checksum>>8) & 0xFF);
    chksum[1] = (unsigned char)(checksum & 0xFF);

    buf[42]=chksum[0];
    buf[43]=chksum[1];
    //for new checksum

pcap_dump((u_char*)pd, phdr, pkt );
place++;
count++;
}
else {
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}
}
}

```

```

//insert at the end
//replacing block function3 with srtp authentication with checksum
void replace_block_withsrtpauth_end (u_char *pd, const struct pcap_pkthdr *phdr,
const u_char *pkt){
    int p, t=0; u16 *data=new u16[226];
    MRef <SRtpPacket *> packet;

    u16 *src_add=new u16[8];
    u16 *dst_add=new u16[8];
    u16 checksum;
    if (count==place & place<=128){
        list<MRef<setVecotr *>> ::iterator k;
        u_char * buf=(u_char *)pkt;

        unsigned char *bl_vector=new unsigned char[phdr->len];

        for( k = block.begin(),p=1; k!= block.end(); k++,p++ ){
            if (place==p){
                bl_vector=(*k)->getblock();
                // for(int i=0;i<phdr->len;i++) //if full packet is replaced
                for(int i=44;i<phdr->len;i++)
                    buf[i]=bl_vector[i];
            }
        }
        buf[42]=0;
        buf[43]=0;
        packet=SRtpPacket::readPacketF1(buf,phdr->len);
        char *dt=new char[packet->size()];
        dt=packet->getBytes();
        for(int i=36,j=0;i<packet->size();i++,j++){
            data[j]=dt[i];
            t++;
        }
        printf("\n\n");
        for(int i=28,j=0;i<32;i++,j++){
            src_add[j]=dt[i]; //convert unsigned char * to unsigned short
        }
        for(int i=32,j=0;i<36;i++,j++){
            dst_add[j]=dt[i];
        }
        //unsigned short *p = static_cast<unsigned
short*>(static_cast<void*>(&packetBuffer[1]));

        checksum=computeUDPChecksum(190, src_add, dst_add, t, data);
        // u_char *chksum=static_cast<u16>(static_cast<void*>(&checksum));
        u_char *chksum=new u_char[4];

```

```

//converts checksum fro unsigned short to unsigned char*

    chksum[0] = (unsigned char)((checksum>>8) & 0xFF);
    chksum[1] = (unsigned char)(checksum & 0xFF);

    buf[42]=chksum[0];
    buf[43]=chksum[1];
    //for new checksum

pcap_dump((u_char*)pd, phdr, pkt );    place++;
count++;

}
else {
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}
}

//This function replace the contend of the packet without replacing sequece no.'
// (okokokok)

void replace_content_srtp (u_char *pd, const struct pcap_pkthdr *phdr, const u_char
*pkt){

    unsigned short *src_add=new unsigned short[8];
    unsigned short *dst_add=new unsigned short[8];
    unsigned short *data=new unsigned short[226];
    unsigned short checksum;
    int t=0;
    if (count==place){

        int p_len=phdr->len;

        /* in buf only encrypted payloads entered*/
        u_char *buf=(u_char*)pkt;
        for (int i=48;i<p_len;i++)//only sequence no. is not modified
        //for (int i=56;i<p_len-4;i++) //only data is modified
            buf[i]='d';
        buf[42]=0;
        buf[43]=0;

        MRef <SRtpPacket *> packet;

        packet=SRtpPacket::readPacketF1(buf,phdr->len);
        buf=packet->getContent();
    }
}

```

```

for(int i=36,j=0;i<p_len;i++,j++){
    data[j]=buf[i];
    t++;
}
for(int i=28,j=0;i<32;i++,j++){
    src_add[j]=pkt[i]; //convert unsigned char * to unsigned short
}
for(int i=32,j=0;i<36;i++,j++){
    dst_add[j]=pkt[i];
}
//unsigned short *p = static_cast<unsigned
short*>(static_cast<void*>(&packetBuffer[1]));
//checksum=udp_sum_calc(190, src_add, dst_add, 1, data);
checksum=computeUDPChecksum(190, src_add, dst_add, t, data);
//printf("%d\n",checksum);
u_char *chksum=new u_char[4];

//converts checksum fro unsigned short to unsigned char*

chksum[0] = (unsigned char)((checksum>>8) & 0xFF);
chksum[1] = (unsigned char)(checksum & 0xFF);

buf[42]=chksum[0];
buf[43]=chksum[1];

pcap_dump((u_char*)pd, phdr, pkt );
count++;
}
else {
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}
}

//replacing block function3 with srtp authentication
void replace_block_withoutsrtpauth (u_char *pd, const struct pcap_pkthdr *phdr,
const u_char *pkt){
    int p;

    if (count==place & place<=128){
        list<MRef<setVecotr *>> ::iterator k;
        u_char *buf=new u_char[phdr->len];
        buf=(u_char *)pkt;

        unsigned char *bl_vector=new unsigned char[phdr->len];

```

```

for( k = block.begin(),p=1; k!= block.end(); k++,p++ ){
    if (place==p){
        bl_vector=(*k)->getblock();
        MRef <SRtpPacket *> packet;

        for (int i=27, j=0;i<phdr->len;i++,j++)
            // for (int i=36;i<phdr->len;i++)
                buf[i]=bl_vector[i];
        packet=SRtpPacket::readPacketF1(buf,phdr->len);
        buf=packet->getContent();
        for(int i=0;i<phdr->len;i++)
            printf("%x ",buf[i]);
            printf("\n\n");
        }
    }

    pcap_dump((u_char*)pd, phdr, pkt );
    place++;
    count++;
}
else if(place>128){
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}
else {
    pcap_dump((u_char*)pd, phdr, pkt );
    count++;
}
}

void mainattacker::RunAttacker(){

    pcap_t *pcap_filefromwright;
    char pcapErr[PCAP_ERRBUF_SIZE];
    int choice;

    char frtcp1[]="/home/morshed/Desktop/Lea/wireshark/srtp_forged4.libpcap";
    // char frtcp1[]="/home/morshed/Desktop/Lea/wireshark/UDP-echo-request-
    example.pcap";

    rblock();
    pcap_filefromwright= pcap_open_offline(frtcp1, pcapErr);

    if (pcap_filefromwright == NULL)
    {
        fprintf(stderr, "pcap_open_offline failed: %s\n", pcapErr);
        exit(EXIT_FAILURE);
    }
}

```

```

//replace_block_callback ();
pd=pcap_dump_open(pcap_filefromwright, firtcp);
cout<<"Enter Your Choice:: ";
cin>>choice;
switch(choice){
    case 1:
        /* Replace all packet of a session with the packet of other session*/
        pcap_loop(pcap_filefromwright, 0, pcap_dump, (u_char *)pd);
        cout<<"Full Session has been replaced by the conversation of other
session."<<endl;
        break;

    case 2:
        /* Insert a packet or block of packet of a session with the packet or block of
packets
* of other session in the front, middle, or end of the captured session. (without
SRTP authentication) */
        cout<<"Enter the position after the packet the forged packet or block will be
inserted (type digit):: "<<endl;
        cin>>place;
        pcap_loop(pcap_filefromwright, 0, insert_block_packet, (u_char *)pd);
        cout<<" Modified data has been inserted after packet no. %d."<<place<<endl;
        break;

    case 3:
        /* Insert a packet or block of packet of a session with the packet or block of
packets
* of other session in the front, middle, or end of the captured session. (with
SRTP authentication) */
        cout<<"Enter the position after the packet the forged packet or block will be
inserted (type digit):: "<<endl;
        cin>>place;
        pcap_loop(pcap_filefromwright, 0, insert_block_packet_Srtp, (u_char *)pd);
        cout<<" Modified data has been inserted after packet no. %d."<<place<<endl;
        break;

    case 4:
        /* Replace the content of a packet of a session with the packet or block of
packets
* of other session in the front, middle, or end of the captured session. (without
SRTP authentication) */
        cout<<"Enter the position after the packet the forged packet or block will be
inserted (type digit):: "<<endl;
        cin>>place;
        pcap_loop(pcap_filefromwright, 0, replace_content_packet, (u_char *)pd);
        cout<<" Modified data has been inserted after packet no. %d."<<place<<endl;
        break;

```

```

case 5:
    /* Replace the content of a packet of a session with the packet or block of
    packets
    * of other session in the front, middle, or end of the captured session. (
    * with SRTP authentication and sequence no.) */
    cout<<"Enter the position after the packet the forged packet or block will be
    inserted (type digit):: "<<endl;
    cin>>place;
    pcap_loop(pcap_filefromwright, 0, replace_content_srtp, (u_char *)pd);
    cout<<" Modified data has been inserted after packet no. %d."<<place<<endl;
    break;

case 6:
    /* Replace the content of a packet of a session with the packet or block of
    packets
    * of other session in the front, middle, or end of the captured session. (
    * without SRTP authentication and sequence no.) */
    cout<<"Enter the position after the packet the forged packet or block will be
    inserted (type digit):: "<<endl;
    cin>>place;
    pcap_loop(pcap_filefromwright, 0, replace_data, (u_char *)pd);
    cout<<" Modified data has been inserted after packet no. %d."<<place<<endl;
    break;

case 7:
    /* Replace the content of a packet of a session with the packet or block of
    packets
    * of other session in the front, middle, or end of the captured session. (
    * without SRTP authentication and sequence no.) */
    cout<<"Enter the position after the packet the forged packet or block will be
    inserted (type digit):: "<<endl;
    cin>>place;
    //pcap_loop(pcap_filefromwright, 0, replace_block_withsrtpauth, (u_char
    *)pd);
    pcap_loop(pcap_filefromwright, 0, replace_block_withsrtpauth_end, (u_char
    *)pd);
    cout<<" Modified data has been inserted after packet no. %d."<<place<<endl;
    break;

default:
    cout<<"Invalid Input"<<endl;
}
//pcap_loop(pcap_filefromwright, 15, pcap_dump, (u_char *)pd);//replace_data
//pcap_loop(pcap_filefromwright, 0, pv_callback, (u_char *)pd);// this line is for
replacing the a block a packet
//pcap_loop(pcap_filefromwright, 0, pblock_callback, (u_char *)pd);//
//pcap_loop(pcap_filefromwright, 0, pc_callback, (u_char *)pd);// this line for
replacing the content of the packet

```



```

//pcap_loop(pcap_filefromwright, 0, pcps_callback, (u_char *)pd);// this line for
replacing the content of the packet without replacing sequence no. without srtp
authentication
//pcap_loop(pcap_filefromwright, 0, pcas_callback, (u_char *)pd);// this line for
replacing the content of the packet without replacing sequence no. with srtp
authentication
// pcap_loop(pcap_filefromwright, 0, psrtp_auth_callback, (u_char *)pd);// this line
for replacing the content of the packet replacing sequence no. with srtp authentication
//pcap_loop(pcap_filefromwright, 0, bsrtp_auth_callback, (u_char *)pd);//
(block)this line for replacing the content of the packet replacing sequence no. with
srtp authentication
// pcap_loop(pcap_filefromwright, 0, replace_block_withoutsrtpauth, (u_char
*)pd);// (block)this line for replacing the content of the packet replacing sequence no.
without srtp authentication
// pcap_loop(pcap_filefromwright, 0, replace_block_withsrtpauth, (u_char *)pd);//
(block)this line for replacing the content of the packet replacing sequence no. with
srtp authentication
pcap_close(pcap_filefromwright);
}

/*Create RTP packet*/
RtpPacket *RtpPacket::readPacketF1(unsigned char *buf, int pkt_length){
#define UDP_SIZE 65536
int j;
uint8_t cc;
int i=pkt_length;

if( i < 0 ){
/**#ifdef DEBUG_OUTPUT
merror("recvfrom:");
#endif*/
return NULL;
}

if( i < 12 ){
/* too small to contain an RTP header */
return NULL;
}

cc = buf[0] & 0x0F;
if( i < 12 + cc * 4 ){
/* too small to contain an RTP header with cc CSRC */
return NULL;
}

RtpHeader hdr;
hdr.setVersion( ( buf[0] >> 6 ) & 0x03 );

```

```

hdr.setExtension( ( buf[0] >> 4 ) & 0x01 );
hdr.setCSRCCount( cc );
hdr.setMarker( ( buf[1] >> 7 ) & 0x01 );
hdr.setPayloadType( buf[1] & 0x7F );
hdr.setSeqNo( ( ((uint16_t)buf[2]) << 8 ) | buf[3] );
cerr << "GOT SEQN" << hdr.getSeqNo() << endl;

int tmp = *((int*)(buf + 4));
tmp = ntohs32(tmp);
hdr.setTimestamp(tmp);

tmp = *((int*)(buf + 8));
tmp = ntohs32(tmp);
hdr.setSSRC(tmp);

for( j = 0 ; j < cc ; j++ ) {
    tmp = *((int*)(buf + 12 + j*4));
    tmp = ntohs32(tmp);
    hdr.setSSRC(tmp);
}
int datalen = i - 12 - cc*4;

RtpPacket * rtp = new RtpPacket(hdr, (unsigned char *)&buf[12+4*cc], datalen);
return rtp;
}

/*Create SRTP packet*/
SRtpPacket *SRtpPacket::readPacketF1(unsigned char *pkt, int pkt_length){

#define UDP_SIZE 65536
    int i, j;
    for (j=0;j<pkt_length;j++)
        buf[j]=pkt[j];
    if( pkt_length < 0 ){
/*#ifdef DEBUG_OUTPUT
        merror("recvfrom:");
#endif*/
        return NULL;
    }

    return readPacketF2(buf, pkt_length);
}

SRtpPacket *SRtpPacket::readPacketF2(byte_t *buf, unsigned buflen) {
#define UDP_SIZE 65536
    uint8_t j;
    uint8_t cc;

```

```

if( buflen < 12 ){
    /* too small to contain an RTP header */
    return NULL;
}

    cc = buf[0] & 0x0F;
if( buflen < 12 + cc * 4 ){
    /* too small to contain an RTP header with cc CCSRC */
    return NULL;
}

RtpHeader hdr;
hdr.setVersion( ( buf[0] >> 6 ) & 0x03 );
hdr.setExtension( ( buf[0] >> 4 ) & 0x01 );
hdr.setCSRCCount( cc );
hdr.setMarker( ( buf[1] >> 7 ) & 0x01 );
hdr.setPayloadType( buf[1] & 0x7F );

hdr.setSeqNo( ( ((uint16_t)buf[2]) << 8 ) | buf[3] );
hdr.setTimestamp( U32_AT( buf + 4 ) );
hdr.setSSRC( U32_AT( buf + 8 ) );

for( j = 0 ; j < cc ; j++ )
    hdr.addCSRC( U32_AT( buf + 12 + j*4 ) );

int datalen = buflen - 12 - cc*4;
unsigned char *data = (unsigned char *)&buf[ 12 + 4*cc ];
SRtpPacket *srtp = new SRtpPacket( hdr, data, datalen, NULL, 0, NULL, 0 );

return srtp;
}

```

E. Required Time to derive Session Keys (SRTP packet/micro second)

1 st Test	2 nd Test	3 rd Test	4 th Test
00:00:00.000136	00:00:00.000147	00:00:00.000124	00:00:00.000140
00:00:00.000132	00:00:00.000148	00:00:00.000119	00:00:00.000133
00:00:00.000138	00:00:00.000139	00:00:00.000123	00:00:00.000149
00:00:00.000132	00:00:00.000143	00:00:00.000119	00:00:00.000132
00:00:00.000138	00:00:00.000142	00:00:00.000123	00:00:00.000137
00:00:00.000131	00:00:00.000132	00:00:00.000118	00:00:00.000132
00:00:00.000138	00:00:00.000136	00:00:00.000122	00:00:00.000137
00:00:00.000132	00:00:00.000142	00:00:00.000119	00:00:00.000132
00:00:00.000136	00:00:00.000147	00:00:00.000136	00:00:00.000137
00:00:00.000132	00:00:00.000133	00:00:00.000119	00:00:00.000141
00:00:00.000131	00:00:00.000132	00:00:00.000118	00:00:00.000144
00:00:00.000132	00:00:00.000133	00:00:00.000118	00:00:00.000144
00:00:00.000133	00:00:00.000132	00:00:00.000118	00:00:00.000132
00:00:00.000143	00:00:00.000132	00:00:00.000118	00:00:00.000132
00:00:00.000145	00:00:00.000135	00:00:00.000120	00:00:00.000134
00:00:00.000134	00:00:00.000134	00:00:00.000119	00:00:00.000132
00:00:00.000133	00:00:00.000146	00:00:00.000118	00:00:00.000133
00:00:00.000143	00:00:00.000144	00:00:00.000118	00:00:00.000132
00:00:00.000132	00:00:00.000132	00:00:00.000119	00:00:00.000376
00:00:00.000131	00:00:00.000133	00:00:00.000119	00:00:00.000133
00:00:00.000132	00:00:00.000132	00:00:00.000118	00:00:00.000142
00:00:00.000135	00:00:00.000142	00:00:00.000123	00:00:00.000137
00:00:00.000135	00:00:00.000133	00:00:00.000119	00:00:00.000140
00:00:00.000140	00:00:00.000137	00:00:00.000124	00:00:00.000150
00:00:00.000132	00:00:00.000133	00:00:00.000120	00:00:00.000134
00:00:00.000146	00:00:00.000137	00:00:00.000124	00:00:00.000137
00:00:00.000133	00:00:00.000131	00:00:00.000119	00:00:00.000132
00:00:00.000138	00:00:00.000148	00:00:00.000123	00:00:00.000137
00:00:00.000132	00:00:00.000142	00:00:00.000118	00:00:00.000134
00:00:00.000137	00:00:00.000138	00:00:00.000124	00:00:00.000138
00:00:00.000133	00:00:00.000132	00:00:00.000168	00:00:00.000133
00:00:00.000139	00:00:00.000138	00:00:00.000142	00:00:00.000137
00:00:00.000133	00:00:00.000132	00:00:00.000134	00:00:00.000158
00:00:00.000138	00:00:00.000136	00:00:00.000157	00:00:00.000139
00:00:00.000140	00:00:00.000132	00:00:00.000134	00:00:00.000187
00:00:00.000148	00:00:00.000147	00:00:00.000312	00:00:00.000153
00:00:00.000146	00:00:00.000133	00:00:00.000137	00:00:00.000134
00:00:00.000140	00:00:00.000137	00:00:00.000144	00:00:00.000139
00:00:00.000145	00:00:00.000131	00:00:00.000134	00:00:00.000134
00:00:00.000144	00:00:00.000132	00:00:00.000133	00:00:00.000145
00:00:00.000135	00:00:00.000130	00:00:00.000133	00:00:00.000133

00:00:00.000133	00:00:00.000132	00:00:00.000133	00:00:00.000133
00:00:00.000134	00:00:00.000132	00:00:00.000133	00:00:00.000138
00:00:00.000133	00:00:00.000143	00:00:00.000146	00:00:00.000147
00:00:00.000132	00:00:00.000143	00:00:00.000144	00:00:00.000176
00:00:00.000147	00:00:00.000133	00:00:00.000146	00:00:00.000144
00:00:00.000132	00:00:00.000132	00:00:00.000134	00:00:00.000134
00:00:00.000131	00:00:00.000131	00:00:00.000132	00:00:00.000166
00:00:00.000137	00:00:00.000137	00:00:00.000148	00:00:00.000144
00:00:00.000131	00:00:00.000131	00:00:00.000133	00:00:00.000134
00:00:00.000130	00:00:00.000136	00:00:00.000137	00:00:00.000152
00:00:00.000134	00:00:00.000141	00:00:00.000132	00:00:00.000192
00:00:00.000137	00:00:00.000135	00:00:00.000137	00:00:00.000149
00:00:00.000131	00:00:00.000134	00:00:00.000144	00:00:00.000133
00:00:00.000134	00:00:00.000140	00:00:00.000148	00:00:00.000143
00:00:00.000130	00:00:00.000136	00:00:00.000132	00:00:00.000134
00:00:00.000136	00:00:00.000142	00:00:00.000137	00:00:00.000136
00:00:00.000133	00:00:00.000135	00:00:00.000133	00:00:00.000132
00:00:00.000146	00:00:00.000140	00:00:00.000138	00:00:00.000149
00:00:00.000147	00:00:00.000134	00:00:00.000133	00:00:00.000134
00:00:00.000148	00:00:00.000157	00:00:00.000139	00:00:00.000148
00:00:00.000131	00:00:00.000134	00:00:00.000146	00:00:00.000131
00:00:00.000136	00:00:00.000137	00:00:00.000149	00:00:00.000136
00:00:00.000136	00:00:00.000140	00:00:00.000133	00:00:00.000144
00:00:00.000138	00:00:00.000139	00:00:00.000136	00:00:00.000140
00:00:00.000133	00:00:00.000133	00:00:00.000133	00:00:00.000134
00:00:00.000131	00:00:00.000133	00:00:00.000133	00:00:00.000130
00:00:00.000132	00:00:00.000132	00:00:00.000130	00:00:00.000132
00:00:00.000131	00:00:00.000132	00:00:00.000133	00:00:00.000133
00:00:00.000131	00:00:00.000139	00:00:00.000132	00:00:00.000131
00:00:00.000132	00:00:00.000131	00:00:00.000132	00:00:00.000131
00:00:00.000141	00:00:00.000131	00:00:00.000134	00:00:00.000144
00:00:00.000144	00:00:00.000131	00:00:00.000133	00:00:00.000142
00:00:00.000147	00:00:00.000143	00:00:00.000135	00:00:00.000132
00:00:00.000138	00:00:00.000133	00:00:00.000134	00:00:00.000132
00:00:00.000148	00:00:00.000125	00:00:00.000152	00:00:00.000136
00:00:00.000132	00:00:00.000118	00:00:00.000147	00:00:00.000133
00:00:00.000146	00:00:00.000135	00:00:00.000167	00:00:00.000150
00:00:00.000146	00:00:00.000119	00:00:00.000146	00:00:00.000132
00:00:00.000138	00:00:00.000125	00:00:00.000150	00:00:00.000138
00:00:00.000132	00:00:00.000119	00:00:00.000145	00:00:00.000133
00:00:00.000137	00:00:00.000124	00:00:00.000139	00:00:00.000148
00:00:00.000133	00:00:00.000119	00:00:00.000135	00:00:00.000145
00:00:00.000138	00:00:00.000124	00:00:00.000161	00:00:00.000137
00:00:00.000132	00:00:00.000119	00:00:00.000134	00:00:00.000133
00:00:00.000147	00:00:00.000124	00:00:00.000154	00:00:00.000138
00:00:00.000132	00:00:00.000118	00:00:00.000146	00:00:00.000184
00:00:00.000136	00:00:00.000124	00:00:00.000150	00:00:00.000136
00:00:00.000143	00:00:00.000119	00:00:00.000148	00:00:00.000132

00:00:00.000137	00:00:00.000125	00:00:00.000140	00:00:00.000137
00:00:00.000132	00:00:00.000133	00:00:00.000136	00:00:00.000132
00:00:00.000137	00:00:00.000133	00:00:00.000139	00:00:00.000136
00:00:00.000133	00:00:00.000145	00:00:00.000135	00:00:00.000131
00:00:00.000132	00:00:00.000137	00:00:00.000135	00:00:00.000144
00:00:00.000133	00:00:00.000132	00:00:00.000134	00:00:00.000145
00:00:00.000132	00:00:00.000132	00:00:00.000134	00:00:00.000132
00:00:00.000135	00:00:00.000132	00:00:00.000141	00:00:00.000132
00:00:00.000132	00:00:00.000139	00:00:00.000145	00:00:00.000132
00:00:00.000132	00:00:00.000133	00:00:00.000145	00:00:00.000133
00:00:00.000133	00:00:00.000133	00:00:00.000146	00:00:00.00013 3
00:00:00.000133	00:00:00.000132	00:00:00.000134	00:00:00.000131
00:00:00.000133	00:00:00.000142	00:00:00.000141	00:00:00.000131
00:00:00.000143	00:00:00.000132	00:00:00.000133	00:00:00.000143
00:00:00.000146	00:00:00.000132	00:00:00.000133	00:00:00.000143
00:00:00.000149	00:00:00.000136	00:00:00.000139	00:00:00.000139
00:00:00.000132	00:00:00.000131	00:00:00.000133	00:00:00.000132
00:00:00.000138	00:00:00.000136	00:00:00.000139	00:00:00.000148
00:00:00.000133	00:00:00.000132	00:00:00.000143	00:00:00.000144
00:00:00.000138	00:00:00.000132	00:00:00.000147	00:00:00.000136
00:00:00.000132	00:00:00.000131	00:00:00.000134	00:00:00.000133
00:00:00.000136	00:00:00.000135	00:00:00.000136	00:00:00.000138
00:00:00.000133	00:00:00.000130	00:00:00.000132	00:00:00.000132
00:00:00.000138	00:00:00.000136	00:00:00.000137	00:00:00.000138
00:00:00.000142	00:00:00.000141	00:00:00.000133	00:00:00.000132
00:00:00.000148	00:00:00.000148	00:00:00.000151	00:00:00.000149
00:00:00.000144	00:00:00.000131	00:00:00.000144	00:00:00.000142
00:00:00.000142	00:00:00.000137	00:00:00.000148	00:00:00.000137
00:00:00.000133	00:00:00.000131	00:00:00.000134	00:00:00.000132
00:00:00.000137	00:00:00.000136	00:00:00.000136	00:00:00.000149
00:00:00.000132	00:00:00.000131	00:00:00.000133	00:00:00.000134
00:00:00.000132	00:00:00.000132	00:00:00.000131	00:00:00.000131
00:00:00.000131	00:00:00.000144	00:00:00.000133	00:00:00.000131
00:00:00.000131	00:00:00.000144	00:00:00.000133	00:00:00.000131
00:00:00.000142	00:00:00.000133	00:00:00.000131	00:00:00.000144
00:00:00.000145	00:00:00.000133	00:00:00.000145	00:00:00.000157
00:00:00.000141	00:00:00.000133	00:00:00.000144	00:00:00.000144
00:00:00.000142	00:00:00.000134	00:00:00.000133	00:00:00.000142
00:00:00.000144	00:00:00.000132	00:00:00.000135	00:00:00.000131
00:00:00.000141	00:00:00.000131	00:00:00.000133	00:00:00.000132
00:00:00.000146	00:00:00.000136	00:00:00.000137	00:00:00.000135
00:00:00.000130	00:00:00.000143	00:00:00.000132	00:00:00.000131
00:00:00.000134	00:00:00.000146	00:00:00.000139	00:00:00.000138
00:00:00.000130	00:00:00.000131	00:00:00.000135	00:00:00.000132
00:00:00.000137	00:00:00.000137	00:00:00.000152	00:00:00.000148
00:00:00.000140	00:00:00.000131	00:00:00.000145	00:00:00.000131
00:00:00.000134	00:00:00.000137	00:00:00.000149	00:00:00.000136
00:00:00.000131	00:00:00.000132	00:00:00.000133	00:00:00.000167

00:00:00.000148	00:00:00.000139	00:00:00.000138	00:00:00.000157
00:00:00.000141	00:00:00.000144	00:00:00.000134	00:00:00.000146
00:00:00.000155	00:00:00.000135	00:00:00.000146	00:00:00.000153
00:00:00.000144	00:00:00.000130	00:00:00.000150	00:00:00.000144
00:00:00.000148	00:00:00.000135	00:00:00.000138	00:00:00.000138
00:00:00.000131	00:00:00.000133	00:00:00.000134	00:00:00.000133
00:00:00.000136	00:00:00.000138	00:00:00.000138	00:00:00.000137
00:00:00.000131	00:00:00.000132	00:00:00.000134	00:00:00.000132
00:00:00.000149	00:00:00.000138	00:00:00.000138	00:00:00.000137
00:00:00.000136	00:00:00.000132	00:00:00.000134	00:00:00.000132
00:00:00.000132	00:00:00.000132	00:00:00.000134	00:00:00.000144
00:00:00.000133	00:00:00.000136	00:00:00.000146	00:00:00.000150
00:00:00.000131	00:00:00.000138	00:00:00.000143	00:00:00.000132
00:00:00.000132	00:00:00.000135	00:00:00.000146	00:00:00.000146
00:00:00.000131	00:00:00.000144	00:00:00.000134	00:00:00.000142
00:00:00.000144	00:00:00.000144	00:00:00.000133	00:00:00.000144
00:00:00.000130	00:00:00.000147	00:00:00.000134	00:00:00.000132
00:00:00.000131	00:00:00.000132	00:00:00.000134	00:00:00.000133
00:00:00.000144	00:00:00.000132	00:00:00.000134	00:00:00.000132
00:00:00.000144	00:00:00.000133	00:00:00.000133	00:00:00.000131
00:00:00.000132	00:00:00.000133	00:00:00.000132	00:00:00.000132
00:00:00.000135	00:00:00.000139	00:00:00.000140	00:00:00.000138
00:00:00.000139	00:00:00.000138	00:00:00.000140	00:00:00.000143
00:00:00.000136	00:00:00.000139	00:00:00.000138	00:00:00.000149
00:00:00.000132	00:00:00.000133	00:00:00.000156	00:00:00.000132
00:00:00.000136	00:00:00.000139	00:00:00.000139	00:00:00.000137
00:00:00.000132	00:00:00.000133	00:00:00.000132	00:00:00.000133
00:00:00.000137	00:00:00.000147	00:00:00.000137	00:00:00.000137
00:00:00.000132	00:00:00.000132	00:00:00.000133	00:00:00.000132
00:00:00.000137	00:00:00.000141	00:00:00.000152	00:00:00.000137
00:00:00.000142	00:00:00.000132	00:00:00.000143	00:00:00.000142
00:00:00.000147	00:00:00.000148	00:00:00.000150	00:00:00.000160
00:00:00.000132	00:00:00.000146	00:00:00.000135	00:00:00.000131
00:00:00.000136	00:00:00.000140	00:00:00.000184	00:00:00.000147
00:00:00.000131	00:00:00.000134	00:00:00.000135	00:00:00.000143
00:00:00.000135	00:00:00.000139	00:00:00.000137	00:00:00.000136
00:00:00.000131	00:00:00.000135	00:00:00.000146	00:00:00.000133
00:00:00.000131	00:00:00.000134	00:00:00.000146	00:00:00.000133
00:00:00.000131	00:00:00.000132	00:00:00.000144	00:00:00.000134
00:00:00.000142	00:00:00.000131	00:00:00.000132	00:00:00.000133
00:00:00.000142	00:00:00.000132	00:00:00.000132	00:00:00.000133
00:00:00.000147	00:00:00.000142	00:00:00.000162	00:00:00.000133
00:00:00.000141	00:00:00.000132	00:00:00.000133	00:00:00.000143
00:00:00.000143	00:00:00.000131	00:00:00.000134	00:00:00.000133
00:00:00.000130	00:00:00.000138	00:00:00.000134	00:00:00.000132
00:00:00.000131	00:00:00.000132	00:00:00.000133	00:00:00.000133
00:00:00.000139	00:00:00.000135	00:00:00.000139	00:00:00.000137
00:00:00.000130	00:00:00.000132	00:00:00.000133	00:00:00.000133

00:00:00.000135	00:00:00.000137	00:00:00.000140	00:00:00.000137
00:00:00.000129	00:00:00.000132	00:00:00.000132	00:00:00.000132
00:00:00.000134	00:00:00.000137	00:00:00.000151	00:00:00.000156
00:00:00.000138	00:00:00.000132	00:00:00.000144	00:00:00.000143
00:00:00.000146	00:00:00.000137	00:00:00.000138	00:00:00.000137
00:00:00.000130	00:00:00.000143	00:00:00.000132	00:00:00.000133
00:00:00.000132	00:00:00.000148	00:00:00.000138	00:00:00.000136
00:00:00.000139	00:00:00.000132	00:00:00.000133	00:00:00.000133
00:00:00.000133	00:00:00.000137	00:00:00.000171	00:00:00.000136
00:00:00.000140	00:00:00.000131	00:00:00.000146	00:00:00.000133
00:00:00.000141	00:00:00.000148	00:00:00.000138	00:00:00.000161
00:00:00.000132	00:00:00.000134	00:00:00.000133	00:00:00.000143
00:00:00.000138	00:00:00.000139	00:00:00.000150	00:00:00.000148
00:00:00.000132	00:00:00.000133	00:00:00.000134	00:00:00.000133
00:00:00.000133	00:00:00.000132	00:00:00.000133	00:00:00.000132
00:00:00.000133	00:00:00.000135	00:00:00.000143	00:00:00.000135
00:00:00.000131	00:00:00.000133	00:00:00.000132	00:00:00.000132
00:00:00.000147	00:00:00.000144	00:00:00.000142	00:00:00.000132
00:00:00.000136	00:00:00.000134	00:00:00.000134	00:00:00.000145
00:00:00.000146	00:00:00.000133	00:00:00.000134	00:00:00.000133
00:00:00.000145	00:00:00.000133	00:00:00.000133	00:00:00.000145
00:00:00.000133	00:00:00.000133	00:00:00.000134	00:00:00.000144
00:00:00.000133	00:00:00.000132	00:00:00.000134	00:00:00.000132
00:00:00.000132	00:00:00.000133	00:00:00.000134	00:00:00.000133
00:00:00.000131	00:00:00.000134	00:00:00.000133	00:00:00.000132
00:00:00.000137	00:00:00.000139	00:00:00.000150	00:00:00.000137
00:00:00.000132	00:00:00.000135	00:00:00.000134	00:00:00.000131
00:00:00.000138	00:00:00.000139	00:00:00.000143	00:00:00.000148
00:00:00.000132	00:00:00.000133	00:00:00.000136	00:00:00.000143
00:00:00.000139	00:00:00.000138	00:00:00.000148	00:00:00.000148
00:00:00.000132	00:00:00.000141	00:00:00.000134	00:00:00.000132
00:00:00.000138	00:00:00.000139	00:00:00.000141	00:00:00.000136
00:00:00.000134	00:00:00.000134	00:00:00.000137	00:00:00.000170
00:00:00.000139	00:00:00.000138	00:00:00.000142	00:00:00.000150
00:00:00.000143	00:00:00.000131	00:00:00.000134	00:00:00.000132
00:00:00.000138	00:00:00.000138	00:00:00.000149	00:00:00.000137
00:00:00.000132	00:00:00.000134	00:00:00.000132	00:00:00.000133
00:00:00.000145	00:00:00.000137	00:00:00.000138	00:00:00.000138
00:00:00.000133	00:00:00.000145	00:00:00.000133	00:00:00.000133
00:00:00.000148	00:00:00.000149	00:00:00.000138	00:00:00.000139
00:00:00.000134	00:00:00.000143	00:00:00.000134	00:00:00.000158
00:00:00.000140	00:00:00.000149	00:00:00.000138	00:00:00.000151
00:00:00.000132	00:00:00.000147	00:00:00.000134	00:00:00.000537
00:00:00.000132	00:00:00.000143	00:00:00.000133	00:00:00.000136
00:00:00.000134	00:00:00.000137	00:00:00.000134	00:00:00.000133
00:00:00.000133	00:00:00.000133	00:00:00.000133	00:00:00.000132
00:00:00.000152	00:00:00.000134	00:00:00.000133	00:00:00.000131
00:00:00.000135	00:00:00.000132	00:00:00.000133	00:00:00.000132

00:00:00.000134	00:00:00.000132	00:00:00.000142	00:00:00.000145
00:00:00.000136	00:00:00.000133	00:00:00.000135	00:00:00.000145
00:00:00.000147	00:00:00.000274	00:00:00.000168	00:00:00.000143
00:00:00.000132	00:00:00.000131	00:00:00.000133	00:00:00.000132
00:00:00.000138	00:00:00.000136	00:00:00.000149	00:00:00.000137
00:00:00.000132	00:00:00.000145	00:00:00.000133	00:00:00.000134
00:00:00.000138	00:00:00.000146	00:00:00.000137	00:00:00.000137
00:00:00.000131	00:00:00.000131	00:00:00.000133	00:00:00.000134
00:00:00.000138	00:00:00.000137	00:00:00.000137	00:00:00.000139
00:00:00.000133	00:00:00.000132	00:00:00.000132	00:00:00.000155
00:00:00.000138	00:00:00.000148	00:00:00.000140	00:00:00.000138
00:00:00.000145	00:00:00.000131	00:00:00.000132	00:00:00.000130
00:00:00.000138	00:00:00.000136	00:00:00.000144	00:00:00.000137
00:00:00.000132	00:00:00.000132	00:00:00.000132	00:00:00.000132
00:00:00.000137	00:00:00.000139	00:00:00.000137	00:00:00.000137
00:00:00.000132	00:00:00.000133	00:00:00.000133	00:00:00.000134
00:00:00.000138	00:00:00.000139	00:00:00.000158	00:00:00.000135
00:00:00.000131	00:00:00.000131	00:00:00.000132	00:00:00.000137
00:00:00.000137	00:00:00.000134	00:00:00.000150	00:00:00.000152
00:00:00.000131	00:00:00.000133	00:00:00.000132	00:00:00.000142
00:00:00.000139	00:00:00.000137	00:00:00.000137	00:00:00.000136
00:00:00.000134	00:00:00.000133	00:00:00.000132	00:00:00.000132
00:00:00.000132	00:00:00.000133	00:00:00.000134	00:00:00.000133
00:00:00.000133	00:00:00.000143	00:00:00.000134	00:00:00.000134
00:00:00.000132	00:00:00.000131	00:00:00.000145	00:00:00.000132
00:00:00.000131	00:00:00.000132	00:00:00.000133	00:00:00.000132
00:00:00.000132	00:00:00.000141	00:00:00.000133	00:00:00.000132
00:00:00.000134	00:00:00.000145	00:00:00.000133	00:00:00.000142
00:00:00.000132	00:00:00.000149	00:00:00.000133	00:00:00.000149
00:00:00.000142	00:00:00.000146	00:00:00.000134	00:00:00.000142
00:00:00.000149	00:00:00.000131	00:00:00.000131	00:00:00.000132
00:00:00.000139	00:00:00.000137	00:00:00.000153	00:00:00.000137
00:00:00.000143	00:00:00.000142	00:00:00.000145	00:00:00.000132
00:00:00.000140	00:00:00.000143	00:00:00.000157	00:00:00.000149
00:00:00.000134	00:00:00.000134	00:00:00.000147	00:00:00.000132
00:00:00.000142	00:00:00.000139	00:00:00.000144	00:00:00.000138
00:00:00.000133	00:00:00.000133	00:00:00.000135	00:00:00.000132
00:00:00.000140	00:00:00.000148	00:00:00.000150	00:00:00.000147
00:00:00.000133	00:00:00.000134	00:00:00.000134	00:00:00.000142
00:00:00.000138	00:00:00.000138	00:00:00.000140	00:00:00.000137
00:00:00.000132	00:00:00.000133	00:00:00.000134	00:00:00.000132
00:00:00.000139	00:00:00.000138	00:00:00.000140	00:00:00.000135
00:00:00.000132	00:00:00.000139	00:00:00.000134	00:00:00.000131
00:00:00.000139	00:00:00.000142	00:00:00.000138	00:00:00.000136
00:00:00.000133	00:00:00.000131	00:00:00.000136	00:00:00.000143
00:00:00.000139	00:00:00.000137	00:00:00.000140	00:00:00.000137
00:00:00.000132	00:00:00.000132	00:00:00.000134	00:00:00.000131
00:00:00.000139	00:00:00.000137	00:00:00.000138	00:00:00.000135

00:00:00.000133	00:00:00.000131	00:00:00.000133	00:00:00.000133
00:00:00.000133	00:00:00.000132	00:00:00.000144	00:00:00.000132
00:00:00.000132	00:00:00.000132	00:00:00.000134	00:00:00.000131
00:00:00.000132	00:00:00.000132	00:00:00.000132	00:00:00.000143
00:00:00.000133	00:00:00.000133	00:00:00.000135	00:00:00.000136
00:00:00.000133	00:00:00.000131	00:00:00.000134	00:00:00.000132
00:00:00.000144	00:00:00.000132	00:00:00.000146	00:00:00.000132
00:00:00.000133	00:00:00.000132	00:00:00.000134	00:00:00.000131
00:00:00.000132	00:00:00.000141	00:00:00.000134	00:00:00.000131
00:00:00.000133	00:00:00.000144	00:00:00.000134	00:00:00.000132
00:00:00.000143	00:00:00.000134	00:00:00.000134	00:00:00.000142
00:00:00.000143	00:00:00.000133	00:00:00.000134	00:00:00.000142
00:00:00.000141	00:00:00.000141	00:00:00.000141	00:00:00.000138
00:00:00.000132	00:00:00.000132	00:00:00.000134	00:00:00.000132
00:00:00.000138	00:00:00.000137	00:00:00.000142	00:00:00.000136
00:00:00.000132	00:00:00.000133	00:00:00.000145	00:00:00.000131
00:00:00.000136	00:00:00.000137	00:00:00.000138	00:00:00.000134
00:00:00.000140	00:00:00.000132	00:00:00.000134	00:00:00.000145
00:00:00.000137	00:00:00.000137	00:00:00.000166	00:00:00.000149
00:00:00.000147	00:00:00.000132	00:00:00.000145	00:00:00.000146
00:00:00.000137	00:00:00.000147	00:00:00.000139	00:00:00.000131
00:00:00.000132	00:00:00.000133	00:00:00.000135	00:00:00.000137
00:00:00.000137	00:00:00.000148	00:00:00.000139	00:00:00.000142
00:00:00.000131	00:00:00.000131	00:00:00.000135	00:00:00.000143
00:00:00.000136	00:00:00.000139	00:00:00.000139	00:00:00.000134
00:00:00.000131	00:00:00.000132	00:00:00.000134	00:00:00.000139
00:00:00.000137	00:00:00.000140	00:00:00.000151	00:00:00.000133
00:00:00.000148	00:00:00.000133	00:00:00.000134	00:00:00.000148
00:00:00.000143	00:00:00.000132	00:00:00.000132	00:00:00.000134
00:00:00.000144	00:00:00.000133	00:00:00.000145	00:00:00.000138
00:00:00.000130	00:00:00.000132	00:00:00.000133	00:00:00.000133
00:00:00.000132	00:00:00.000131	00:00:00.000134	00:00:00.000138
00:00:00.000132	00:00:00.000133	00:00:00.000136	00:00:00.000139
00:00:00.000132	00:00:00.000132	00:00:00.000144	00:00:00.000142
00:00:00.000134	00:00:00.000132	00:00:00.000132	00:00:00.000131
00:00:00.000132	00:00:00.000132	00:00:00.000134	00:00:00.000137
00:00:00.000131	00:00:00.000130	00:00:00.000134	00:00:00.000132
00:00:00.000137	00:00:00.000140	00:00:00.000138	00:00:00.000137
00:00:00.000132	00:00:00.000144	00:00:00.000134	00:00:00.000131
00:00:00.000140	00:00:00.000143	00:00:00.000143	00:00:00.000132
00:00:00.000132	00:00:00.000143	00:00:00.000133	00:00:00.000132
00:00:00.000136	00:00:00.000141	00:00:00.000140	00:00:00.000132
00:00:00.000132	00:00:00.000132	00:00:00.000134	00:00:00.000133
00:00:00.000137	00:00:00.000141	00:00:00.000148	00:00:00.000131
00:00:00.000133	00:00:00.000132	00:00:00.000132	00:00:00.000132
00:00:00.000138	00:00:00.000140	00:00:00.000138	00:00:00.000132
00:00:00.000142	00:00:00.000132	00:00:00.000134	00:00:00.000141
00:00:00.000152	00:00:00.000146	00:00:00.000148	00:00:00.000144

00:00:00.000132	00:00:00.000136	00:00:00.000134	00:00:00.000134
00:00:00.000139	00:00:00.000143	00:00:00.000151	00:00:00.000133
00:00:00.000133	00:00:00.000133	00:00:00.000134	00:00:00.000142
00:00:00.000138	00:00:00.000140	00:00:00.000138	00:00:00.000134
00:00:00.000132	00:00:00.000139	00:00:00.000135	00:00:00.000138
00:00:00.000132	00:00:00.000132	00:00:00.000134	00:00:00.000133
00:00:00.000134	00:00:00.000134	00:00:00.000135	00:00:00.000139
00:00:00.000134	00:00:00.000133	00:00:00.000135	00:00:00.000135
00:00:00.000133	00:00:00.000132	00:00:00.000154	00:00:00.000150
00:00:00.000141	00:00:00.000132	00:00:00.000134	00:00:00.000145
00:00:00.000135	00:00:00.000131	00:00:00.000134	00:00:00.000133
00:00:00.000132	00:00:00.000132	00:00:00.000133	00:00:00.000134
00:00:00.000133	00:00:00.000132	00:00:00.000145	00:00:00.000133
00:00:00.000131	00:00:00.000132	00:00:00.000135	00:00:00.000134
00:00:00.000144	00:00:00.000132	00:00:00.000134	00:00:00.000134
00:00:00.000143	00:00:00.000132	00:00:00.000135	00:00:00.000132
00:00:00.000132	00:00:00.000132	00:00:00.000136	00:00:00.000133
00:00:00.000133	00:00:00.000132	00:00:00.000135	00:00:00.000145
00:00:00.000139	00:00:00.000143	00:00:00.000153	00:00:00.000142
00:00:00.000133	00:00:00.000134	00:00:00.000134	00:00:00.000137
00:00:00.000145	00:00:00.000150	00:00:00.000142	00:00:00.000132
00:00:00.000131	00:00:00.000143	00:00:00.000134	00:00:00.000158
00:00:00.000159	00:00:00.000139	00:00:00.000140	00:00:00.000139
00:00:00.000143	00:00:00.000131	00:00:00.000134	00:00:00.000137
00:00:00.000147	00:00:00.000145	00:00:00.000162	00:00:00.000131
00:00:00.000133	00:00:00.000132	00:00:00.000133	00:00:00.000137
00:00:00.000137	00:00:00.000147	00:00:00.000139	00:00:00.000153
00:00:00.000131	00:00:00.000131	00:00:00.000135	00:00:00.000143
00:00:00.000136	00:00:00.000137	00:00:00.000141	00:00:00.000133
00:00:00.000133	00:00:00.000128	00:00:00.000135	00:00:00.000137
00:00:00.000138	00:00:00.000138	00:00:00.000140	00:00:00.000134
00:00:00.000142	00:00:00.000142	00:00:00.000136	00:00:00.000138
00:00:00.000136	00:00:00.000136	00:00:00.000153	00:00:00.000145
00:00:00.000132	00:00:00.000132	00:00:00.000134	00:00:00.000148
00:00:00.000132	00:00:00.000133	00:00:00.000133	00:00:00.000132
00:00:00.000132	00:00:00.000132	00:00:00.000133	00:00:00.000138
00:00:00.000131	00:00:00.000132	00:00:00.000134	00:00:00.000132
00:00:00.000131	00:00:00.000133	00:00:00.000132	00:00:00.000132
00:00:00.000141	00:00:00.000132	00:00:00.000133	00:00:00.000134
00:00:00.000131	00:00:00.000132	00:00:00.000133	00:00:00.000131
00:00:00.000132	00:00:00.000144	00:00:00.000152	00:00:00.000198
00:00:00.000131	00:00:00.000132	00:00:00.000135	00:00:00.000148
00:00:00.000131	00:00:00.000132	00:00:00.000134	00:00:00.000132
00:00:00.000135	00:00:00.000137	00:00:00.000138	00:00:00.000131
00:00:00.000130	00:00:00.000132	00:00:00.000133	00:00:00.000138
00:00:00.000139	00:00:00.000139	00:00:00.000140	00:00:00.000132
00:00:00.000144	00:00:00.000319	00:00:00.000133	00:00:00.000132
00:00:00.000146	00:00:00.000139	00:00:00.000137	00:00:00.000134

00:00:00.000131	00:00:00.000131	00:00:00.000132	00:00:00.000134
00:00:00.000136	00:00:00.000137	00:00:00.000138	00:00:00.000133
00:00:00.000131	00:00:00.000131	00:00:00.000133	00:00:00.000141
00:00:00.000135	00:00:00.000137	00:00:00.000137	00:00:00.000135
00:00:00.000130	00:00:00.000133	00:00:00.000132	00:00:00.000132
00:00:00.000147	00:00:00.000136	00:00:00.000137	00:00:00.000133
00:00:00.000131	00:00:00.000131	00:00:00.000132	00:00:00.000131
00:00:00.000136	00:00:00.000136	00:00:00.000138	00:00:00.000144
00:00:00.000132	00:00:00.000132	00:00:00.000132	00:00:00.000143
00:00:00.000137	00:00:00.000138	00:00:00.000138	00:00:00.000132
00:00:00.000130	00:00:00.000132	00:00:00.000133	00:00:00.000133
00:00:00.000130	00:00:00.000132	00:00:00.000144	00:00:00.000139
00:00:00.000143	00:00:00.000132	00:00:00.000133	00:00:00.000133
00:00:00.000147	00:00:00.000288	00:00:00.000133	00:00:00.000145
00:00:00.000143	00:00:00.000134	00:00:00.000133	00:00:00.000131
00:00:00.000143	00:00:00.000134	00:00:00.000135	00:00:00.000159
00:00:00.000133	00:00:00.000134	00:00:00.000146	00:00:00.000143
00:00:00.000133	00:00:00.000127	00:00:00.000133	00:00:00.000147
00:00:00.000132	00:00:00.000123	00:00:00.000134	00:00:00.000133
00:00:00.000132	00:00:00.000123	00:00:00.000135	00:00:00.000137
00:00:00.000185	00:00:00.000124	00:00:00.000135	00:00:00.000131
00:00:00.000132	00:00:00.000122	00:00:00.000135	00:00:00.000136
00:00:00.000137	00:00:00.000128	00:00:00.000148	00:00:00.000133
00:00:00.000132	00:00:00.000126	00:00:00.000133	00:00:00.000138
00:00:00.000139	00:00:00.000127	00:00:00.000140	00:00:00.000142
00:00:00.000138	00:00:00.000121	00:00:00.000139	00:00:00.000136
00:00:00.000148	00:00:00.000128	00:00:00.000149	00:00:00.000132
00:00:00.000143	00:00:00.000125	00:00:00.000133	00:00:00.000132
00:00:00.000136	00:00:00.000128	00:00:00.000138	00:00:00.000132
00:00:00.000131	00:00:00.000122	00:00:00.000133	00:00:00.000131
00:00:00.000137	00:00:00.000131	00:00:00.000138	00:00:00.000131
00:00:00.000132	00:00:00.000122	00:00:00.000142	00:00:00.000141
00:00:00.000143	00:00:00.000126	00:00:00.000142	00:00:00.000131
00:00:00.000131	00:00:00.000118	00:00:00.000134	00:00:00.000132
00:00:00.000136	00:00:00.000123	00:00:00.000138	00:00:00.000131
00:00:00.000144	00:00:00.000119	00:00:00.000133	00:00:00.000131
00:00:00.000146	00:00:00.000125	00:00:00.000139	00:00:00.000135
00:00:00.000133	00:00:00.000132	00:00:00.000135	00:00:00.000130
00:00:00.000137	00:00:00.000143	00:00:00.000150	00:00:00.000139
00:00:00.000133	00:00:00.000143	00:00:00.000145	00:00:00.000144
00:00:00.000131	00:00:00.000119	00:00:00.000133	00:00:00.000146
00:00:00.000131	00:00:00.000119	00:00:00.000134	00:00:00.000131
00:00:00.000131	00:00:00.000118	00:00:00.000133	00:00:00.000136
00:00:00.000143	00:00:00.000121	00:00:00.000134	00:00:00.000131
00:00:00.000143	00:00:00.000123	00:00:00.000134	00:00:00.000135
00:00:00.000132	00:00:00.000122	00:00:00.000132	00:00:00.000130
00:00:00.000132	00:00:00.000125	00:00:00.000133	00:00:00.000147
00:00:00.000132	00:00:00.000122	00:00:00.000145	00:00:00.000131

00:00:00.000130	00:00:00.000122	00:00:00.000142	00:00:00.000136
00:00:00.000136	00:00:00.000128	00:00:00.000137	00:00:00.000137
00:00:00.000141	00:00:00.000119	00:00:00.000132	00:00:00.000132
00:00:00.000145	00:00:00.000128	00:00:00.000158	00:00:00.000146
00:00:00.000147	00:00:00.000122	00:00:00.000139	00:00:00.000136
00:00:00.000146	00:00:00.000128	00:00:00.000137	00:00:00.000143
00:00:00.000131	00:00:00.000121	00:00:00.000131	00:00:00.000133
00:00:00.000136	00:00:00.000130	00:00:00.000137	00:00:00.000140
00:00:00.000132	00:00:00.000122	00:00:00.000153	00:00:00.000139
00:00:00.000135	00:00:00.000129	00:00:00.000143	00:00:00.000132
00:00:00.000131	00:00:00.000123	00:00:00.000133	00:00:00.000134
00:00:00.000146	00:00:00.000128	00:00:00.000137	00:00:00.000133
00:00:00.000143	00:00:00.000123	00:00:00.000134	00:00:00.000132
00:00:00.000136	00:00:00.000129	00:00:00.000138	00:00:00.000132
00:00:00.000131	00:00:00.000122	00:00:00.000145	00:00:00.000131
00:00:00.000135	00:00:00.000128	00:00:00.000148	00:00:00.000132
00:00:00.000130	00:00:00.000120	00:00:00.000132	00:00:00.000132
00:00:00.000136	00:00:00.000128	00:00:00.000138	00:00:00.000132
00:00:00.000142	00:00:00.000123	00:00:00.000132	00:00:00.000132
00:00:00.000131	00:00:00.000123	00:00:00.000132	00:00:00.000132
00:00:00.000130	00:00:00.000122	00:00:00.000134	00:00:00.000132
00:00:00.000131	00:00:00.000122	00:00:00.000131	00:00:00.000132
00:00:00.000131	00:00:00.000123	00:00:00.000198	00:00:00.000143
00:00:00.000130	00:00:00.000123	00:00:00.000148	00:00:00.000134
00:00:00.000147	00:00:00.000123	00:00:00.000132	00:00:00.000150
00:00:00.000130	00:00:00.000124	00:00:00.000131	00:00:00.000143
00:00:00.000141	00:00:00.000146	00:00:00.000132	00:00:00.000139
00:00:00.000142	00:00:00.000123	00:00:00.000131	00:00:00.000131
00:00:00.000135	00:00:00.000133	00:00:00.000137	00:00:00.000145
00:00:00.000132	00:00:00.000123	00:00:00.000133	00:00:00.000132
00:00:00.000149	00:00:00.000142	00:00:00.000164	00:00:00.000147
00:00:00.000131	00:00:00.000123	00:00:00.000134	00:00:00.000131
00:00:00.000137	00:00:00.000128	00:00:00.000149	00:00:00.000137
00:00:00.000131	00:00:00.000122	00:00:00.000144	00:00:00.000128
00:00:00.000147	00:00:00.000142	00:00:00.000138	00:00:00.000138
00:00:00.000143	00:00:00.000122	00:00:00.000133	00:00:00.000142
00:00:00.000136	00:00:00.000127	00:00:00.000163	00:00:00.000136
00:00:00.000132	00:00:00.000122	00:00:00.000137	00:00:00.000132
00:00:00.000137	00:00:00.000142	00:00:00.000142	00:00:00.000133
00:00:00.000131	00:00:00.000124	00:00:00.000134	00:00:00.000132
00:00:00.000136	00:00:00.000147	00:00:00.000140	00:00:00.000132
00:00:00.000144	00:00:00.000121	00:00:00.000145	00:00:00.000133
00:00:00.000147	00:00:00.000127	00:00:00.000148	00:00:00.000132
00:00:00.000132	00:00:00.000121	00:00:00.000133	00:00:00.000132
00:00:00.000164	00:00:00.000128	00:00:00.000138	00:00:00.000144
00:00:00.000143	00:00:00.000140	00:00:00.000133	00:00:00.000132
00:00:00.000134	00:00:00.000123	00:00:00.000134	00:00:00.000132
00:00:00.000132	00:00:00.000122	00:00:00.000133	00:00:00.000137

00:00:00.000131	00:00:00.000122	00:00:00.000132	00:00:00.000143
00:00:00.000132	00:00:00.000122	00:00:00.000133	00:00:00.000132
00:00:00.000131	00:00:00.000123	00:00:00.000134	00:00:00.000135
00:00:00.000144	00:00:00.000126	00:00:00.000144	00:00:00.000133
00:00:00.000142	00:00:00.000123	00:00:00.000145	00:00:00.000140
00:00:00.000131	00:00:00.000124	00:00:00.000134	00:00:00.000136
00:00:00.000132	00:00:00.000123	00:00:00.000133	00:00:00.000139
00:00:00.000132	00:00:00.000124	00:00:00.000133	00:00:00.000122
00:00:00.000131	00:00:00.000137	00:00:00.000133	00:00:00.000142
00:00:00.000138	00:00:00.000131	00:00:00.000180	00:00:00.000122
00:00:00.000143	00:00:00.000123	00:00:00.000133	00:00:00.000127
00:00:00.000130	00:00:00.000132	00:00:00.000139	00:00:00.000122
00:00:00.000136	00:00:00.000123	00:00:00.000133	00:00:00.000142
00:00:00.000132	00:00:00.000127	00:00:00.000137	00:00:00.000124
00:00:00.000137	00:00:00.000138	00:00:00.000134	00:00:00.000147
00:00:00.000135	00:00:00.000140	00:00:00.000155	00:00:00.000121
00:00:00.000135	00:00:00.000133	00:00:00.000133	00:00:00.000127
00:00:00.000123	00:00:00.000133	00:00:00.000149	00:00:00.000121
00:00:00.000126	00:00:00.000132	00:00:00.000142	00:00:00.000128
00:00:00.000123	00:00:00.000133	00:00:00.000149	00:00:00.000140
00:00:00.000124	00:00:00.000134	00:00:00.000134	00:00:00.000123
00:00:00.000123	00:00:00.000144	00:00:00.000137	00:00:00.000155
00:00:00.000124	00:00:00.000145	00:00:00.000133	00:00:00.000133
00:00:00.000133	00:00:00.000134	00:00:00.000136	00:00:00.000149
00:00:00.000133	00:00:00.000133	00:00:00.000132	00:00:00.000142
00:00:00.000180	00:00:00.000133	00:00:00.000133	00:00:00.000149
00:00:00.000133	00:00:00.000133	00:00:00.000143	00:00:00.000136
00:00:00.000139	00:00:00.000180	00:00:00.000133	00:00:00.000132
00:00:00.000133	00:00:00.000133	00:00:00.000147	00:00:00.000137

F. Cumulative Percentage of the delays beyond the minimum required time to generate session keys

<i>Bin</i>	<i>Frequency</i>	<i>Cumulative %</i>
128	0	0.00%
129	1	0.20%
130	0	0.20%
131	69	13.75%
132	95	32.42%
133	49	42.04%
134	16	45.19%
135	18	48.72%
136	35	55.60%
137	27	60.90%
138	32	67.19%
139	16	70.33%
140	0	70.33%
141	45	79.17%
142	19	82.91%
143	31	89.00%
144	30	94.89%
145	7	96.27%
146	13	98.82%
147	3	99.41%
148	2	99.80%
149	1	100.00%
150	0	100.00%

G. Required Time to decrypt SRTP packet (SRTP packet/micro second)

1 st Test	2 nd Test	3 rd Test	4 th Test
0:00:00.000024	00:00:00.000024	00:00:00.000025	00:00:00.000019
00:00:00.000018	00:00:00.000019	00:00:00.000019	00:00:00.000019
00:00:00.000019	00:00:00.000019	00:00:00.000020	00:00:00.000018
00:00:00.000018	00:00:00.000019	00:00:00.000019	00:00:00.000018
00:00:00.000018	00:00:00.000020	00:00:00.000019	00:00:00.000019
00:00:00.000019	00:00:00.000029	00:00:00.000020	00:00:00.000019
00:00:00.000019	00:00:00.000020	00:00:00.000019	00:00:00.000019
00:00:00.000018	00:00:00.000030	00:00:00.000020	00:00:00.000019
00:00:00.000018	00:00:00.000031	00:00:00.000019	00:00:00.000018
00:00:00.000019	00:00:00.000019	00:00:00.000019	00:00:00.000020
00:00:00.000018	00:00:00.000020	00:00:00.000029	00:00:00.000020
00:00:00.000018	00:00:00.000030	00:00:00.000020	00:00:00.000019
00:00:00.000019	00:00:00.000020	00:00:00.000020	00:00:00.000019
00:00:00.000018	00:00:00.000019	00:00:00.000020	00:00:00.000019
00:00:00.000019	00:00:00.000019	00:00:00.000020	00:00:00.000019
00:00:00.000019	00:00:00.000020	00:00:00.000021	00:00:00.000018
00:00:00.000019	00:00:00.000019	00:00:00.000020	00:00:00.000020
00:00:00.000019	00:00:00.000019	00:00:00.000019	00:00:00.000033
00:00:00.000020	00:00:00.000020	00:00:00.000021	00:00:00.000019
00:00:00.000018	00:00:00.000021	00:00:00.000020	00:00:00.000019
00:00:00.000019	00:00:00.000021	00:00:00.000020	00:00:00.000020
00:00:00.000019	00:00:00.000034	00:00:00.000020	00:00:00.000019
00:00:00.000020	00:00:00.000020	00:00:00.000027	00:00:00.000018
00:00:00.000020	00:00:00.000021	00:00:00.000020	00:00:00.000019
00:00:00.000019	00:00:00.000031	00:00:00.000021	00:00:00.000019
00:00:00.000019	00:00:00.000033	00:00:00.000020	00:00:00.000020
00:00:00.000019	00:00:00.000019	00:00:00.000025	00:00:00.000021
00:00:00.000025	00:00:00.000040	00:00:00.000029	00:00:00.000021
00:00:00.000020	00:00:00.000020	00:00:00.000019	00:00:00.000034
00:00:00.000019	00:00:00.000020	00:00:00.000027	00:00:00.000020
00:00:00.000020	00:00:00.000020	00:00:00.000020	00:00:00.000021
00:00:00.000019	00:00:00.000019	00:00:00.000020	00:00:00.000031
00:00:00.000019	00:00:00.000020	00:00:00.000019	00:00:00.000033
00:00:00.000019	00:00:00.000019	00:00:00.000020	00:00:00.000019
00:00:00.000019	00:00:00.000020	00:00:00.000019	00:00:00.000040
00:00:00.000019	00:00:00.000019	00:00:00.000019	00:00:00.000019
00:00:00.000018	00:00:00.000020	00:00:00.000019	00:00:00.000018
00:00:00.000018	00:00:00.000020	00:00:00.000019	00:00:00.000018
00:00:00.000019	00:00:00.000020	00:00:00.000019	00:00:00.000019
00:00:00.000020	00:00:00.000024	00:00:00.000020	00:00:00.000018
00:00:00.000019	00:00:00.000020	00:00:00.000019	00:00:00.000019

H. Cumulative Percentage of the delays beyond the minimum required time to decrypt an SRTP packet

<i>Bin</i>	<i>Frequency</i>	<i>Cumulative %</i>
16	0	0.00%
17	1	0.20%
18	138	27.36%
19	280	82.48%
20	18	86.02%
21	18	89.57%
22	22	93.90%
23	29	99.61%
24	1	99.80%
25	0	99.80%
26	0	99.80%
27	0	99.80%
28	0	99.80%
29	1	100.00%
30	0	100.00%
More	0	100.00%

I. CPU used for performance Evaluation

```

processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 15
model name    : Intel(R) Core(TM)2 CPU          T5500 @
1.66GHz
stepping      : 2
cpu MHz       : 1000.000
cache size    : 2048 KB
physical id   : 0
siblings      : 2
core id       : 0
cpu cores     : 2
apicid        : 0
initial apicid : 0
fdiv_bug      : no
hlt_bug       : no
f00f_bug      : no
coma_bug      : no
fpu           : yes
fpu_exception : yes
cpuid level   : 10
wp            : yes
flags         : fpu vme de pse tsc msr pae mce cx8 apic
sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr
sse sse2 ss ht tm pbe nx lm constant_tsc arch_perfmon
pebs bts pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16
xtpr pdcm lahf_lm tpr_shadow
bogomips      : 3324.79
clflush size  : 64
power management:

processor      : 1
vendor_id     : GenuineIntel
cpu family    : 6
model         : 15
model name    : Intel(R) Core(TM)2 CPU          T5500 @
1.66GHz
stepping      : 2
cpu MHz       : 1000.000
cache size    : 2048 KB
physical id   : 0
siblings      : 2
core id       : 1
cpu cores     : 2

```

