# Security Implementation of a Real-time Transport Protocol Based Application (TranSecure)






*Developed by*

**Junaid Aslam**
(24-CS/MS/01)

**Saad Rafique**
(15-CS/MS/01)

*Supervised by*

**Dr. S. Tauseef-ur-Rehman**
Head Department of Telecommunication and Computer Engineering

Department of Computer Sciences
International Islamic University, Islamabad
(2004)

# International Islamic University
# Islamabad

18-7-2004
June ~~21,~~ 2004

## Final Approval

It is certified that we have read the project titled "**Security Implementation of a Real-time Transport Protocol based Application (TranSecure)**" submitted by Junaid Aslam and Saad Rafique. It is our judgment that this project is of sufficient standard to warrant its acceptance by International Islamic University for the Master's degree in Computer Sciences.

## Committee

**External Examiner**
Prof. Dr. M. Qasim Rind
Preston University

**Internal Examiner**
Prof. Dr. Sikandar Hayat Khayal
Head, Faculty of Applied Sciences
International Islamic University

Mr. M [redacted]

**Supervisor**
Prof. Dr. S. Tauseef-ur-Rehman
Head, Department of Telecommunication
International Islamic University

A DISSERTATION SUBMITTED TO THE

DEPARTMENT OF COMPUTER SCIENCE

INTERNATIONAL ISALMIC UNIVERSITY, ISLAMABAD

AS

A PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE AWARD OF DEGREE OF

MASTERS OF COMPUTER SCIENCE

# DECLARATION

We, hereby declare that the project titled "Security Implementation of a Real-time Transport Protocol based Application (TranSecure)", neither as a whole nor as a part thereof has been copied out from any source. It is further declared that we have developed this application and the accompanied report entirely on the basis of our personal efforts made under the guidance of the kind supervisor. No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute, if found we will stand responsible.

**Junaid Aslam**

24-CS/MS/01

**Saad Rafique**

15-CS/MS/01

# ACKNOWLEDGEMENTS

First of all we are thankful to Almighty ALLAH (SWT) most compassionate and most merciful, without his blessings completion of this project was not possible.

Then is the place of our respected supervisor Dr. S. Tauseef-ur-Rehman. His all time appreciation has been the motivating force behind the successful completion of this application. He has been very kind and extra ordinarily cooperative to us during the whole process of development.

We are also thankful to our friends in Masters of Computer Sciences who have been a source of support, encouragement and motivation during the whole academic and development period.

Above all we owe this success of ours to our loving parents without whose pure prayers, true encouragements, moral, and financial support we would never have succeeded.

**Junaid Aslam**
24-CS/MS/01

**Saad Rafique**
15-CS/MS/01

# PROJECT IN BRIEF

| | |
|---|---|
| **Project Title** | Security Implementation of RTP based Application |
| **Objective** | To develop a Real Time Transport Protocol based application that can securely transmit data over the network. |
| **Undertaken By** | **Junaid Aslam**<br>(24-CS/MS/00)<br>**Saad Rafique**<br>(15-CS/MS/00) |
| **Supervised By** | **Dr. S. Tauseef-ur-Rehman**<br>Head Department of Telecommunication and Computer Engineering<br>International Islamic University, Islamabad |
| **Protocol Level Technologies** | Real Time Transport Protocol |
| **Programming Languages** | JAVA |
| **Operating System/ Server** | Windows 2000 professional / Windows XP Professional |
| **Date Started** | January, 2003 |
| **Date Completed** | April, 2004 |

# Table of Contents


| Content# | Contents | Page# |
|---|---|---|
| 1 | Introduction | 1 |
| 1.2 | TranSecure | 2 |
| 1.3 | Encryption | 2 |
| 1.3.1 | Security Methodologies | 3 |
| 1.3.2 | Analog Scrambling | 3 |
| 1.3.3 | Digital Encryption | 3 |
| 1.3.4 | Signal Encryption | 4 |
| 1.4 | Recommended Security Algorithms | 4 |
| 1.4.1 | DES | 4 |
| 1.4.2 | Triple DES | 6 |
| 1.4.3 | AES | 7 |
| 1.5 | Real-time Transport Protocol | 9 |
| 1.5.1 | RTP Services | 10 |
| 1.5.2 | RTP Architecture | 11 |
| 1.5.3 | Data Packets | 13 |
| 1.5.4 | Control Packets | 15 |
| 1.5.5 | RTP Applications | 16 |
| 1.5.6 | Network Organization | 16 |
| 1.6 | Desired Security Features | 18 |
| 1.6.1 | Confidentiality | 18 |
| 1.6.2 | Integrity and Authenticity | 18 |
| 1.6.3 | Performance Considerations of Security Features | 18 |
| 1.7 | Security Features Provided by RTP | 19 |
| 1.7.1 | Confidentiality | 19 |
| 1.7.2 | Authentication and Integrity | 20 |
| 1.7.3 | Key Management | 20 |
| 1.8 | Introduction to SRTP | 20 |
| 1.8.1 | Security in SRTP | 21 |
| 1.9 | JMF | 22 |
| 1.9.1 | JMF Architecture | 22 |
| 1.9.2 | High Level Architecture | 23 |
| 1.9.3 | Understanding the JMF-RTP API | 24 |
| 1.10 | Working with Real-time Media Streams | 26 |
| 1.10.1 | Streaming Media | 26 |
| 1.10.2 | Protocols for Streaming Media | 26 |
| 1.10.3 | Transmitting Media Streams across the Network | 27 |
| 1.10.4 | Receiving Media Streams from the Network | 28 |
| 2 | Existing System | 29 |
| 2.1 | Proposed System | 29 |
| 2.1.1 | Project Definition | 29 |
| 2.1.2 | Project Scope | 30 |
| 2.1.3 | Efficiency | 30 |
| 2.1.4 | Reliability | 31 |
| 2.1.5 | User Friendliness | 31 |
| 2.1.6 | Future Enhancements | 31 |
| 2.1.7 | Advantages of the Proposed System | 31 |
| 2.2 | System Analysis | 32 |
| 2.2.1 | Object | 32 |
| 2.2.2 | Criteria for Objects | 33 |
| 2.2.2.1 | Retained Information | 33 |
| 2.2.2.2 | Needed Services | 34 |
| 2.2.2.3 | Multiple Attributes | 34 |
| 2.2.2.4 | Common Attributes | 34 |

| | | |
|---|---|---|
| 2.2.2.5 | Common Operations | 34 |
| 2.2.2.6 | Essential Requirements | 34 |
| 2.2.3 | Selected Objects | 35 |
| 2.3 | Domain Analysis | 35 |
| 2.3.1 | Domain to be Investigated | 36 |
| 2.3.1.1 | Interested Domain | 36 |
| 2.3.1.2 | Object Oriented Items | 36 |
| 2.3.1.3 | Non Object Oriented Items | 38 |
| 3 | System Design | 39 |
| 3.1 | Important Classes | 39 |
| 3.2 | Use Cases | 41 |
| 3.2.1 | Actors of the System | 41 |
| 3.2.2 | Domains in our System | 42 |
| 3.2.3 | Use Cases of Transmitter | 42 |
| 3.2.4 | Use Cases of Receiver | 44 |
| 3.2.5 | Use Cases in Extended Format | 46 |
| 3.2.6 | Extended Use Cases of Transmitter | 46 |
| 3.2.7 | Extended Use Cases of Receiver | 51 |
| 3.3 | Use Case Diagrams | 55 |
| 3.4 | Sequence Diagrams | 58 |
| 3.5 | Object Diagrams | 64 |
| 4 | Description of Classes | 66 |
| 4.1 | Implementation | 74 |
| 4.2 | Java | 74 |
| 4.3 | JMF | 75 |
| 4.3.1 | Session Manager | 77 |
| 4.4 | Code Sample for Transmitting and Receiving Audio and Video of Client using RTP | 79 |
| 4.4.1 | Classes and Packages for Media Streaming and Media Reception | 79 |
| 4.4.2 | Classes and Packages for Encryption | 79 |
| 4.4.3 | Configuring a Processor | 82 |
| 4.4.4 | Retrieving the Processor Output | 84 |
| 4.4.5 | Creating a RTP Player for Each New Received Stream | 87 |
| 4.4.6 | Implementing Controller Listener | 90 |
| 4.4.7 | Displaying Media Interface Components | 91 |
| 4.4.8 | Displaying a Visual Component | 92 |
| 4.4.9 | Displaying a Control Panel Component | 92 |
| 5 | Testing | 93 |
| 5.1 | Testing Strategies | 93 |
| 5.1.1 | Specification Testing | 94 |
| 5.1.2 | Black Box Testing | 94 |
| 5.1.3 | White Box Testing | 94 |
| 5.1.4 | Regression Testing | 94 |
| 5.1.5 | Acceptance Testing | 95 |
| 5.1.6 | Assertion Testing | 95 |
| 5.1.7 | Unit Testing | 95 |
| 5.1.8 | System Testing | 95 |
| 5.2 | System Evaluation | 96 |
| 5.3 | Testing TranSecure | 96 |
| 5.4 | Test Results | 97 |
| 6 | Introduction | 101 |
| 6.1 | User Manual for Transmitter | 101 |
| 6.1.1 | Add Receiving Client | 101 |
| 6.1.2 | Remove Receiving Client | 102 |
| 6.1.3 | Media Locator | 104 |
| 6.1.4 | Encrypting Payload | 105 |
| 6.1.5 | Start Transmission | 106 |

6.1.6 Stop Transmission 107
6.2 User Manual for Receiver 108
6.2.1 Add Target Sender 108
6.2.2 Remove Target Sender 109
6.2.3 Decrypt Data 110
6.2.4 Start Receiver 111
6.2.5 Stop Receiver 112

# List of Tables


| Table# | Contents | Page# |
|---|---|---|
| 4.1 | Description of Class TranSecureServ | 66 |
| 4.2 | Description of Class AVTransmit3 | 67 |
| 4.3 | Description of Class RTPSocketAdapter | 68 |
| 4.4 | Description of Class SocketOutputStream :: RTPSocketAdapter | 70 |
| 4.5 | Description of Class SocketInputStream :: RTPSocketAdapter | 70 |
| 4.6 | Description of Class Config | 71 |
| 4.7 | Description of Class Target | 71 |
| 4.8 | Description of Class CipherSelect | 72 |
| 4.9 | Description of Class AVReceive3 | 72 |
| 5.1 | Test Table for 10 KB Test File | 97 |
| 5.2 | Test Table for 100 KB Test File | 98 |
| 5.3 | Test Table for 1 M Test File | 99 |
| 5.4 | Comparison Table of Different Algorithms | 100 |

# List of Figures


| Figure# | Name of Figure | Page# |
|---|---|---|
| 1.1 | Working of DES | 5 |
| 1.2 | Working of Triple DES | 7 |
| 1.3 | Working of AES | 8 |
| 1.4 | RTP Architecture | 9 |
| 1.5 | RTP Running Over Different Protocols | 12 |
| 1.6 | RTP Data Packet Header Format | 13 |
| 1.7 | RTP Network Organization | 17 |
| 1.8 | Recording, Processing and Presenting Time-Based Media | 23 |
| 1.9 | High Level JMF Architecture | 24 |
| 1.10 | RTP Reception | 25 |
| 1.11 | RTP Transmission | 25 |
| 3.1 | Use Case Diagram of Transmitter | 56 |
| 3.2 | Use Case Diagram of Receiver | 57 |
| 3.3 | Sequence Diagram for Add Target Use Case | 58 |
| 3.4 | Sequence Diagram for Remove Target Use Case | 59 |
| 3.5 | Sequence Diagram for Encrypt Data Use Case | 60 |
| 3.6 | Sequence Diagram for Start Transmitter Use Case | 61 |
| 3.7 | Sequence Diagram for Start Receiver Use Case | 62 |
| 3.8 | Sequence Diagram for Decrypt Data Use Case | 63 |
| 3.9 | Object Diagram of Transmitter | 64 |
| 3.10 | Object Diagram of Receiver | 65 |
| 4.1 | Events in JMF | 78 |
| 5.1 | Time Graph for 10 KB Test File | 97 |
| 5.2 | Time Graph for 100 KB Test File | 98 |
| 5.3 | Time Graph for 1 MB Test File | 99 |
| 5.4 | Comparison Graph for Different Algorithms | 100 |
| 6.1 | Add Receiving Client | 101 |
| 6.2 | Remove Receiving Client | 102 |
| 6.3 | After Pressing the "Remove Target" Button this Form will Appear | 103 |
| 6.4 | Media Locator | 104 |
| 6.5 | Encrypting Payload | 105 |
| 6.6 | Start Transmission | 106 |
| 6.7 | Stop Transmission | 107 |
| 6.8 | Add Target Sender | 108 |
| 6.9 | Remove Target Sender | 109 |
| 6.10 | Remove Target | 109 |
| 6.11 | Decrypt Data | 110 |
| 6.12 | Start Receiver | 111 |
| 6.13 | Stop Receiver | 112 |

# CHAPTER 1

# INTRODUCTION

# 1. Introduction

An increasing number of commercial organizations are now coming to realize that information is valuable and needs to be protected from disclosure to unauthorized parties. However, communications security is still relatively new to the commercial sector, and many organizations are uncertain about the options available and how to proceed. This real-time transport protocol based application is intended to assist organizations which are considering implementing a secure communications network.

Communications security is the protection of information during transmission. Information may need to be protected from modification (accidental or malicious), destruction or disclosure. Realistically, it is often not possible to prevent access to transmissions, so information can only be kept secure by disguising the content of the transmission. Modern communications require this to be achieved electronically, usually by some form of scrambling or encryption.

There are many reasons communications security may be required - in a recent survey, the most commonly cited reasons for securing transmissions were the prevention of accidental security breaches, the prevention of purposeful breaches, meeting Privacy Act requirements, and meeting customer expectations [1].

Communications security is common sense. Information is a valuable commodity, and needs to be protected just like other commodities. Protecting your communications is sensible business practice, regardless of whether or not your information is actually 'secret'. Most organizations keep documents in filing cabinets and lock office doors overnight, and communications security is the next obvious step. Secure communications also allow the best use to be made of your communications equipment - for example, fax machines are both faster and cheaper than couriers, and it is easiest to discuss strategies or conduct business negotiations in real time, using a telephone.

## 1.2 TranSecure

TranSecure is a real-time transport protocol based application that enables you to securely transmit real time data such as voice and video over different existing networks i.e.

1. IP/UDP
2. ATM/AAL5
3. IPX

You can choose your own security algorithm from a list of highly secure algorithms i.e.

1. Data Encryption Standard (DES)
2. Triple Data Encryption Standard (3DES)
3. Advanced Encryption Standard (AES)

You also have the option to send the data without any encryption (Null Cipher).

## 1.3 Encryption

Encryption is the process of transforming information from an unsecured form ("clear" or "plaintext") into coded information ("cipher text"), which cannot be easily read by outside parties. The transformation process is controlled by an algorithm and a key. The process must be reversible so that the intended recipient can return the information to its original, readable form, but reversing the process without the appropriate encryption information should be impossible. This means that details of the key must also be kept secret.

Encryption is generally regarded as the safest method of guarding against accidental or purposeful security breaches. The strength of the encryption method is often measured in terms of work factor - the amount of force that is required to 'break' the encryption. A

strong system will take longer to break, although this can be reduced by applying greater force (the more effort that is put into the attack, the less time required to break the code).

### 1.3.1 Security Methodologies

Communications can be protected in a number of ways, and each method has different strengths and weaknesses. Two of the most common techniques are scrambling and encryption. These terms are often used interchangeably, but technically the processes are quite different - one deals with the information being transmitted and the other modifies characteristics of the signal being used for the transmission.

### 1.3.2 Analog Scrambling

Scramblers disguise the transmission signal in a variety of ways. Usually this involves splitting and mixing the frequency spectrum, and scramblers are most commonly used to protect analog voice communications. Most scramblers do not employ keys, so unscrambling the message is relatively straightforward. Scrambling systems provide an intermediate level of security that is effective against casual interception, but can result in a poor voice quality at the receiver.

### 1.3.3 Digital Encryption

Encryptors manipulate the information or data being carried by the signal. They use an algorithm to alter the message content prior to transmission, and normally operate on digital information. For this reason, encryptors are primarily used to secure data transmissions.

Digital encryption can also be used to protect voice transmissions, but this requires the analog voice signal to be converted to a digital signal before the encryption process can

begin. Depending on the method used, this can also impair the quality of the voice signal after decryption. Because the process uses modems and usually requires synchronization, communications may fail over poor quality lines.

### 1.3.4 Signal Encryption

A third approach is signal encryption, which uses an algorithm to manipulate the analog signal. This method is the core of the Safe encryption process, and provides a level of security between scrambling and digital encryption. Safe does not require synchronization, so communications are possible in areas with poor telecommunications infrastructure; the process also ensures good quality in the voice reconstruction.

## 1.4 Recommended Security Algorithms:

Our research deals primarily with Real Time Applications (see section 1.5). For this RFC 3550 has recommended use of the following three algorithms for RTP security:

1. DES
2. Triple DES
3. AES

### 1.4.1 DES

The Data Encryption Standard (DES) cryptographic algorithm was approved in the 1970s as a US federal standard for use on unclassified government communications and is a de-facto standard for modern encryption [2].

The DES algorithm is based on a 128-bit block algorithm developed in the 1960s by IBM. In technical terms, LUCIFER is an iterative block cipher, using Fiestel rounds - a block of data is encrypted a number of several times, each time applying the key to half of the block and then XOR'ing with the other half of the block.

DES was designed to use a 64-bit key to encrypt and decrypt 64-bit blocks of data using a cycle of permutations, swaps, and substitutions. Encryption and decryption use the same key.

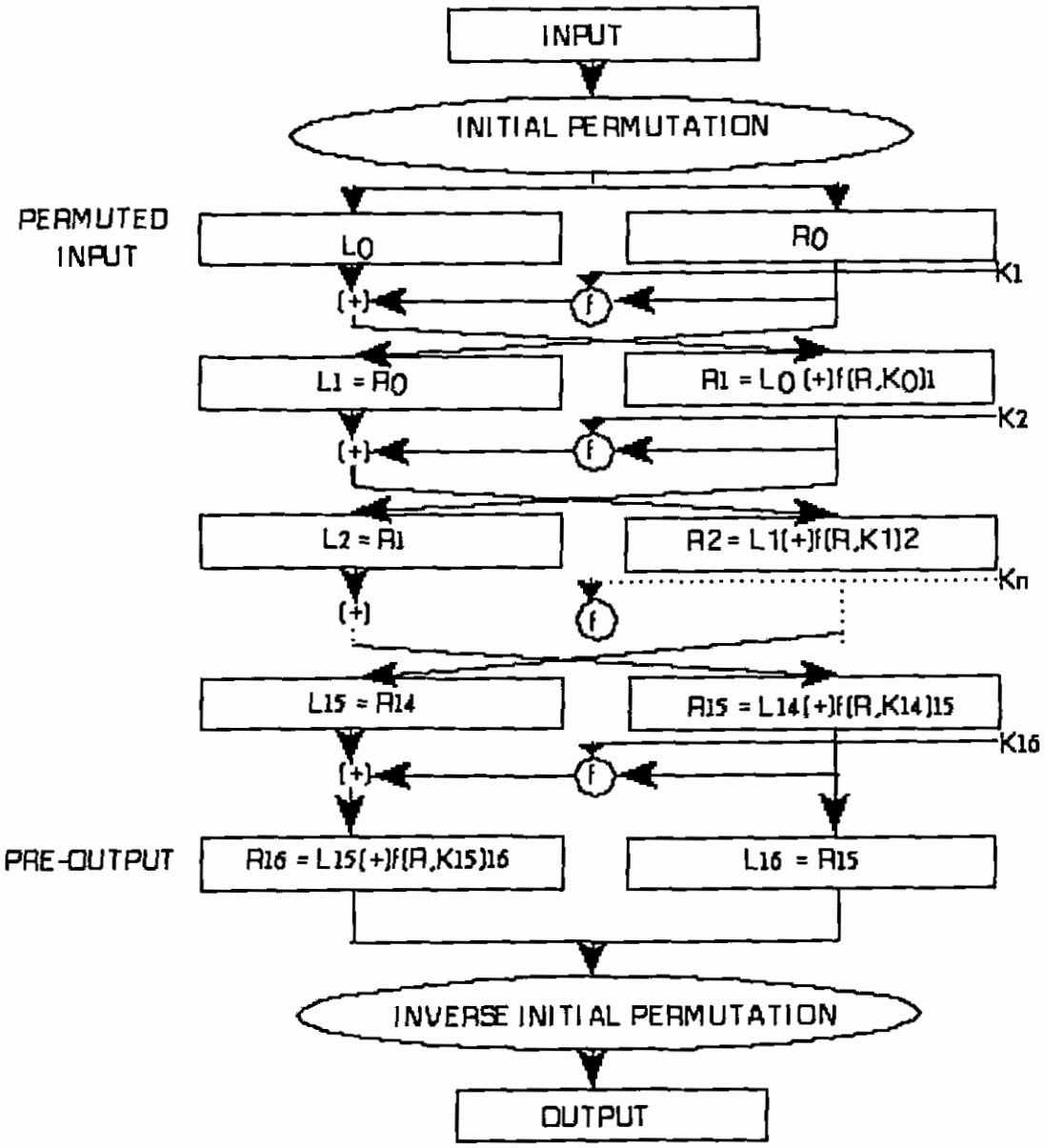


**Figure 1.1: Working of DES**

A block to be encrypted is subjected to an initial permutation, then to a key-dependent computation, and then to a final permutation. The initial and final permutations take the

64-bit block and change the position of each bit in a pre-determined manner. The final permutation is the reverse of the initial permutation

A DES key consists of 64 binary digits of which 56 bits are randomly generated and used directly by the algorithm. The other 8 bits, which are not used by the algorithm, are used for error detection. The 8 error detecting bits are set to make the parity of each 8-bit byte of the key odd, i.e., there is an odd number of "1"s in each 8-bit byte.

DES was not only the Federal Standard, but was also widely used as the algorithm of choice for many years [3]. However, it also became the focus of much dissent because the US imposed controls over its export. Furthermore, the academic cryptographic community argued that DES's 56-bit key was too short to withstand a brute-force attack from modern computers. (Moore's Law states that computer power doubles every 18 months, so a key that could withstand a brute-force guessing attack in 1975 could hardly be expected to withstand the same attack a quarter of a century later) [4]

### 1.4.2 Triple DES

Triple DES is a three fold DES encryption in which two 56 bit keys are applied, that means a practical security of 112 bits in strength, which as far we know today should be reasonably secure. 112 bit 3 DES in outer CBC mode offers an equivalent amount of cryptographic security relative to 168 bit 3 DES in the identical mode. The deep crack project has shown that 112 bit 3 DES offers significant resilience to brut-force cryptanalytic methods. 112 bit 3 DES in outer CBC mode offers the same media stream privacy as 168 bits with a less cumbersome key.

The implementation of this strategy using 3 hardware engines adds only a small propagation delay, however in software using the same single engine; it would take more than double the time required.

To make 3 DES even stronger a triple length key must be used (168 bit key). A triple length key is true 3 DES. It increases the cost of attack when applying the easiest known

attacks i.e. exhaustive key attack but triple key will be a burden if application involves short messages with short life and frequent key changes, also that due to certain space complexity cryptanalytic attacks, 168 bit 3 DES offers cryptographic security no better than 112 bit 3 DES [5].

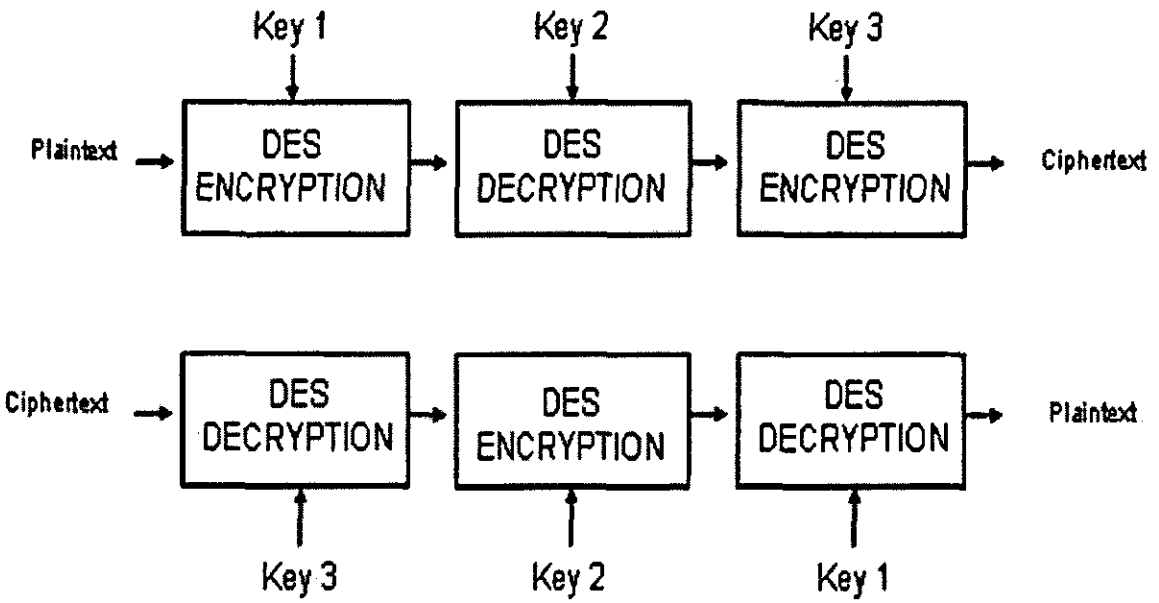


**Figure 1.2: Working of Triple DES**

Moreover a 112 bit key is still quite viable for most commercial applications which require high grade cryptography.

### 1.4.3 AES

The Advanced Encryption Standard supersedes DES as the new information protection standard defined by the US to protect certain levels of Federal information and communications [2]. The selection process for an AES algorithm began in 1997, and the

new standard was finalized in November 2001. AES is a publicly disclosed encryption algorithm, and is unclassified.

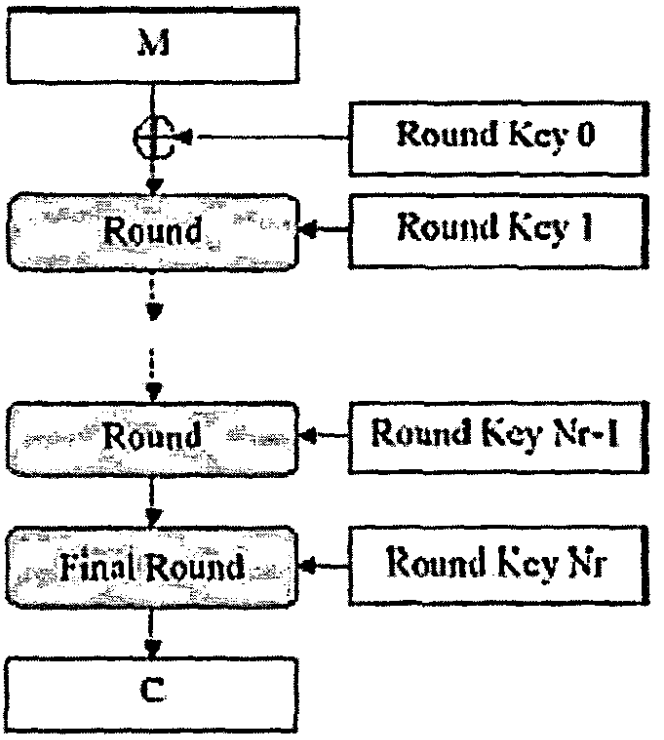


**Figure 1.3: Working of AES**

On January 2, 1997, the US National Institute for Security Technologies (NIST) announced that it was develop an new Advanced Encryption Standard to replace the previous Data Encryption Standard (DES), and called for public submissions for the new AES algorithm [2]. At a minimum, the algorithm would have to implement symmetric key cryptography as a block cipher and support a block size of 128 bits and key sizes of 128, 192, and 256 bits.

## 1.5 Real-Time Transport Protocol

RTP, Real-time Transport Protocol, is an application level protocol that is intended for delivery of delay sensitive content, such as audio and video, through different networks. The purpose of RTP is to facilitate delivery, monitoring, reconstruction, mixing and synchronization of data streams. Although RTP does not provide quality of service on IP networks, its mixers can be used to facilitate multimedia delivery on a wide range of link types and speeds. RTP is designed to use both unicast and multicast transport protocols [6].

Even though RTP is a relatively new protocol, it is widely used by applications like Real Network's RealPlayer, Apple's QuickTime and Microsoft's NetMeeting. Some of the common applications of RTP are audio and video streaming media services and video conferences.

As RTP is usually used through Internet, the network should be considered as insecure. Although many media streams are publicly available, video conferencing use usually requires confidentiality. In many situations it would be preferable if the user could authenticate the originator and ensure the integrity of media streams.

RTP provides end-to-end network delivery services for the transmission of real-time data. RTP is network and transport-protocol independent, though it is often used over UDP.

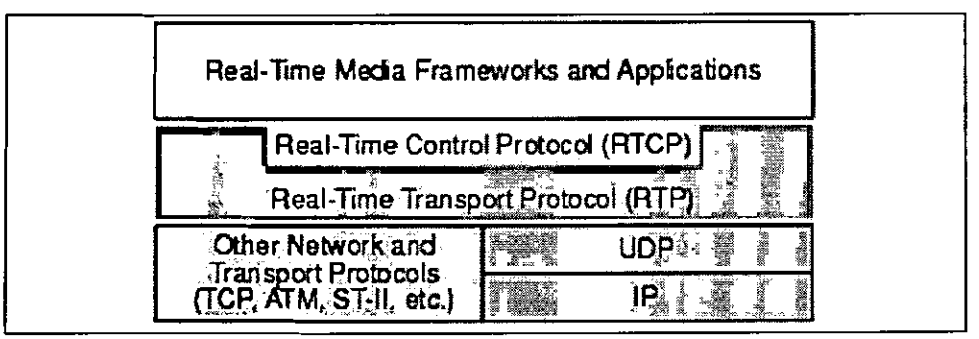


**Figure 1.4: RTP Architecture**

RTP can be used over both unicast and multicast network services. Over a unicast network service, separate copies of the data are sent from the source to each destination. Over a multicast network service, the data is sent from the source only once and the

network is responsible for transmitting the data to multiple locations. Multicasting is more efficient for many multimedia applications, such as video conferences. The standard Internet Protocol (IP) supports multicasting.

### 1.5.1 RTP Services

RTP enables you to identify the type of data being transmitted, determine what order the packets of data should be presented in, and synchronize media streams from different sources.

RTP data packets are not guaranteed to arrive in the order that they were sent. It's up to the receiver to reconstruct the sender's packet sequence and detect lost packets using the information provided in the packet header.

While RTP does not provide any mechanism to ensure timely delivery or provides other quality of service guarantees, it is augmented by a control protocol (RTCP) that enables you to monitor the quality of the data distribution. RTCP also provides control and identification mechanisms for RTP transmissions.

If quality of service is essential for a particular application, RTP can be used over a resource reservation protocol that provides connection-oriented services.

RTP defines the roles for two active application level devices that may reside on the network

1. Mixers
2. Translators

In essence the difference between a translator and a mixer is, that mixers change synchronization source identifiers, whereas translators do not.

Mixers have many similar characteristic features with routers, as they connect two or more networks together. Mixers process RTCP packets and may perform payload format translations. They also perform remixing of RTP streams. The purpose of mixers is to

allow users behind low speed links to receive high speed transmissions by receiving all high speed streams, down-mixing them to one or more lower speed streams and forwarding these low speed streams to receivers. Reverse will of course be done for possible return packets. Mixers have to regenerate timing information and change SSRC's, as they essentially create new streams based on one or more existing streams. Mixers are non transparent devices. As mixers may combine several encrypted streams, they are capable of encrypting and decrypting RTP streams.

Translators are transparent on RTP level – they leave SSRC identifiers intact. The purpose of the translators is to perform payload format conversions, tunneling of the packets through firewalls, adding or removing encryption and enabling the coexistence of the different networking technologies. Whereas mixers could be described as RTP routers, closest equivalent for RTP translators is an application level proxy.

There exist a few limitations for placing multiple translators or mixers on the same node i.e. their network address (on IP network consisting port number and IP address) must be unique and they may not perform same forwarding task unless they have not been partitioned to prevent them from forwarding same packets to same receivers [7].

### 1.5.2 RTP Architecture

An RTP session is an association among a set of applications communicating with RTP. A session is identified by a network address and a pair of ports. One port is used for the media data and the other is used for control (RTCP) data.

A participant is a single machine, host, or user participating in the session. Participation in a session can consist of passive reception of data (receiver), active transmission of data (sender), or both.

Each media type is transmitted in a different session. For example, if both audio and video are used in a conference, one session is used to transmit the audio data and a separate session is used to transmit the video data. This enables participants to choose

which media types they want to receive, for example, someone who has a low-bandwidth network connection might only want to receive the audio portion of a conference.

RTP is a modular protocol. The base protocol is defined by RFC 3550. The usage of RTP for a specific purpose requires that an application area specific RTP profile must be implemented. RFC 3550 defines basic fields for the transportation of real time data. It also defines RTCP, RTP Control protocol, whose purpose is to provide feedback on transmission quality, information about participants of RTP session, and enable minimal session control services.

RTP profiles are used for refining the basic RTP protocol to suit for a particular application area. Commonly RTP profiles refine the meanings of the fields provided by the basic RTP protocol. RTP profiles also add new fields and rules. RTP profiles define how and by which formats data is encapsulated to RTP packets.

In contrary to many protocols, RTP is usually implemented by each application, and not by an operating system or by a separate stack. These implementations may, and often are, based on generic RTP libraries. Existence of the application dependent profiles almost mandates that the RTP service must be implemented on an application basis. RTP protocol is transport independent and it can be used over various networking technologies. The most common transport protocols for RTP are IP/UDP, ATM/AAL5 and IPX. Figure below shows the RTP on the protocol stack. Note that although RTCP can be considered as a protocol running over RTP, it is actually only a special type of RTP packet.

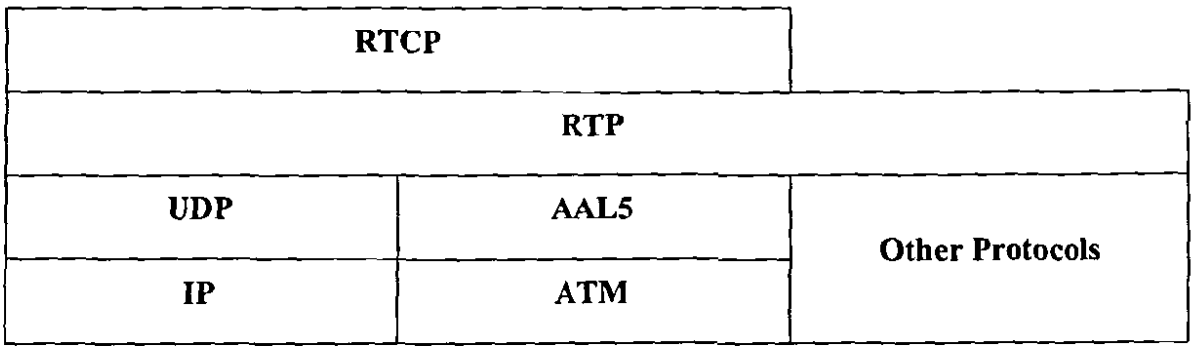


**Figure 1.5: RTP running over different protocols**

It should be understood that the basic RTP has never been intended as a complete protocol, but as a framework for building application protocols. As the consequence, a RTP based system commonly relies on non-RTP protocols to negotiate and establish the sessions. RTP protocol uses synchronization source (SSRC) identifiers as addresses of the peers. SSRCs are unique within the session and are chosen randomly when a participant joins to the session. RTP peers automatically detect and correct SSRC collisions.

### 1.5.3 Data Packets

The media data for a session is transmitted as a series of packets. A series of data packets that originate from a particular source is referred to as an RTP stream. Each RTP data packet in a stream contains two parts, a structured header and the actual data (the packet's payload).

Bit:0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 16 7 8 9 0 1 2 3 4 5 6 7 8 9 0 31

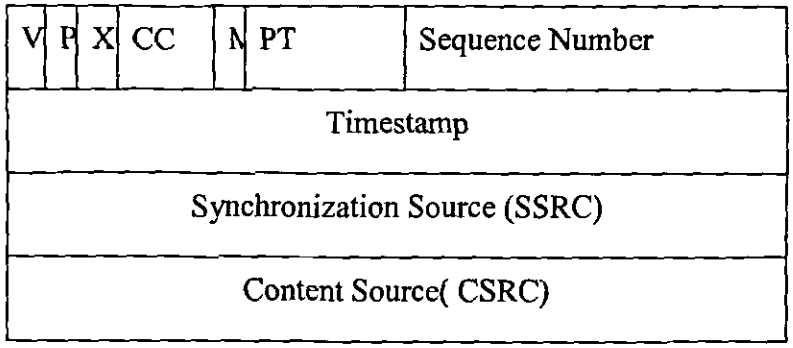


**Figure 1.6: RTP data-packet header format**

The header of an RTP data packet contains:

- **The RTP version number (V)**: 2 bits. The version defined by the current specification is 2.

- **Padding** (P): 1 bit. If the padding bit is set, there are one or more bytes at the end of the packet that are not part of the payload. The very last byte in the packet indicates the number of bytes of padding. The padding is used by some encryption algorithms.

- **Extension** (X): 1 bit. If the extension bit is set, the fixed header is followed by one header extension. This extension mechanism enables implementations to add information to the RTP Header.

- **CSRC Count** (CC): 4 bits. The number of CSRC identifiers that follow the fixed header. If the CSRC count is zero, the synchronization source is the source of the payload.

- **Marker (M)**: 1 bit. A marker bit defined by the particular media profile.

- **Payload Type** (PT): 7 bits. An index into a media profile table that describes the payload format. The payload mappings for audio and video are specified in RFC 3550.

- **Sequence Number**: 16 bits. A unique packet number that identifies this packet's position in the sequence of packets. The packet number is incremented by one for each packet sent.

- **Timestamp**: 32 bits. Reflects the sampling instant of the first byte in the payload. Several consecutive packets can have the same timestamp if they are logically generated at the same time, for example, if they are all part of the same video frame.

- **SSRC**: 32 bits. Identifies the synchronization source. If the CSRC count is zero, the payload source is the synchronization source. If the CSRC count is nonzero, the SSRC identifies the mixer.

- **CSRC**: 32 bits each. Identifies the contributing sources for the payload. The number of contributing sources is indicated by the CSRC count field; there can be

up to 16 contributing sources. If there are multiple contributing sources, the payload is the mixed data from those sources.

### 1.5.4 Control Packets

In addition to the media data for a session, control data (RTCP) packets are sent periodically to all of the participants in the session. RTCP packets can contain information about the quality of service for the session participants, information about the source of the media being transmitted on the data port, and statistics pertaining to the data that has been transmitted so far.

There are several types of RTCP packets:

1. Sender Report
2. Receiver Report
3. Source Description
4. Bye
5. Application-specific

RTCP packets are "stackable" and are sent as a compound packet that contains at least two packets, a report packet and a source description packet.

All participants in a session send RTCP packets. A participant that has recently sent data packets issues a sender report. The sender report (SR) contains the total number of packets and bytes sent as well as information that can be used to synchronize media streams from different sessions.

Session participants periodically issue receiver reports for all of the sources from which they are receiving data packets. A receiver report (RR) contains information about the number of packets lost, the highest sequence number received, and a timestamp that can be used to estimate the round-trip delay between a sender and the receiver.

The first packet in a compound RTCP packet has to be a report packet, even if no data has been sent or received, in which case, an empty receiver report is sent.

All compound RTCP packets must include a source description (SDES) element that contains the canonical name (CNAME) that identifies the source. Additional information might be included in the source description, such as the source's name, email address, phone number, geographic location, application name, or a message describing the current state of the source [6].

When a source is no longer active, it sends an RTCP BYE packet. The BYE notice can include the reason that the source is leaving the session.

RTCP APP packets provide a mechanism for applications to define and send custom information via the RTP control port.

### 1.5.5 RTP Applications

RTP applications are often divided into those that need to be able to receive data from the network (RTP Clients) and those that need to be able to transmit data across the network (RTP Servers). Some applications do both, for example, conferencing applications capture and transmit data at the same time that they're receiving data from the network.

### 1.5.6 Network Organization

RTP is delivered as unicast, multicast or as both. As most RTP uses have more than two participants, it is preferable that RTP is transported on multicast capable network. Translator can be located in unicast network boundary, where it will replicate packets for unicast participants of RTP sessions [8].

Multiple RTP streams can be down-mixed and combined for slow links. Application level firewall might be bypassed by using two translators with a tunnel though the firewall between them. As mixers are allowed to change Synchronization Source

Identifiers, it is essential that RTP network is loop free (note that underlying network may contain as many loops as it is desired).

A simple RTP network is presented below. If ATM network would run IP over AAL5, no translator would be necessary. As ATM has quality of service, it might be preferable to run RTP without IP.

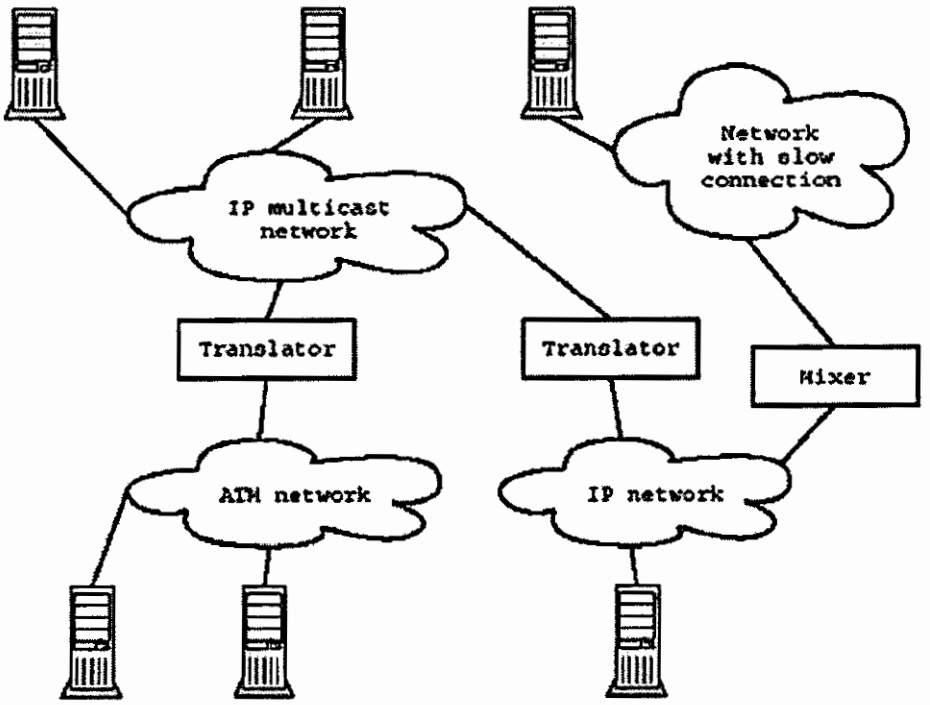


**Figure 1.7: RTP Network Organization**

## 1.6 Desired Security Features

The following are some of the desired security features in a real-time environment.

### 1.6.1 Confidentiality

RTP is commonly used for broadcasting content through network. On such use confidentiality is not necessarily even desired. As RTP is also commonly used for video conferences and for shared white board applications, need for confidentiality should be obvious – for example when RTP based video conference is used for telemedicine, confidentiality is of paramount importance. Even on confidential telemedicine session not all information should be confidential sender and receiver reports containing network performance data can help third party mixers and translators to optimize network usage.

### 1.6.2 Integrity and Authenticity

Even on public broadcast it is useful to be able to verify both the integrity and the origin of the transmission. This information helps the receiver to assess the trustworthiness of the received information. If anyone could modify CNN news broadcast, which many consider reliable information source, results could be potentially disastrous. It should be noted that the possibility for anonymous broadcast is also important, even though not always politically acceptable feature in some countries. It is desired that no participant of the same session could masquerade as some other.

### 1.6.3 Performance Considerations of Security Features

As RTP is often used for transferring huge amounts of time critical data (e.g. video) it is essential that all security features are implemented with minimal delay and jitter. It should be evident that with huge transmission rates even a small timing overhead easily

amounts to huge loss of bandwidth. Although encryption, especially using DES, causes overhead, it is minor compared to CPU requirements of modern compression algorithms for voice and video. Such a small overhead is hardly noticeable, especially as connection speeds available for general population are much lower than the encryption speed of modern desktop computer.

## 1.7 Security Features Provided by RTP

The following are the security features provided by RTP.

### 1.7.1 Confidentiality

The basic RTP protocol, as defined in RFC 3550, provides flexible facilities for encrypting RTP packets. This facility allows splitting packets to encrypted and unencrypted parts, and therefore facilitates the need for unencrypted performance statistics. The default algorithm, which all encryption capable RTP clients must support, is DES-CBC.

Algorithm usage is same as defined in RFC 1423. To prevent known plain text attacks, RTCP headers are obfuscated with 32 bit random prefix. CBC mode has random access property for decryption, which guarantees that the lost packet only prevents decoding of itself and the following packet. This feature is vital, as time sensitive data can usually not be resent [7].

RTP allows the use of any other encryption algorithm, but the algorithm must be negotiated on non RTP means. An application profile may specify additional methods for encrypting the payload. It is suggested by RTP standards that when multicast supporting encryption is offered by the network layer, it should be used instead. In terms on IPsec RTP encryption corresponds using only ESP headers, within pre-established security association.

### 1.7.2 Authentication and Integrity

RTP standard does not specify any authentication, except that implicit authentication is assumed if encryption key is known. RTP assumes that the lower layers of the network will handle more sophisticated authentication. Integrity is verified by sanity checking decrypted headers. Sanity checks verify the known field values, such as protocol version number, packet length and payload type [5].

RTP limits issuing certain commands, like bye (which is used for session termination) to the peers which command affects. This is enforced by checking the SSRC identifiers of the packets containing the commands. This command authentication mechanism only works, if authenticity of RTP stream is ensured by other means.

### 1.7.3 Key management

The only key management feature of RTP is specified in RFC 1890, which specifies a MD5 based method for deriving the encryption key from the password. RTP assumes that more complex key management is either handled by other protocols or by application specific profiles. On conference type applications (video, audio or even only a shared white board) key management is handled by combination of SIP, SAP and SDP protocols. These protocols feature strong authentication and key exchange features, and provide standardized way to establish encrypted conferences using RTP as the transport protocol.

## 1.8 Introduction to SRTP

On occasions, when intense security is needed, RTP applications may take advantage of the fact that new encryption algorithms can be specified dynamically for a session that must be negotiated on non-RTP means, and can consider other, more conventional, protection means for which profiles can be defined. An application profile may specify additional methods for encrypting the payload. A profile defines extensions or

modifications to RTP that are specific to a particular class of applications, services and algorithms that may be offered [5].

Secure Real-time Transport Protocol (SRTP) is an RTP profile that is meant to provide security services like confidentiality, message authentication and replay attack protection to RTP and RTCP traffic. It defines set of default cryptographic transforms and allows introducing new transforms. These transforms assets low computational costs and support bandwidth economy by limiting packet expansion. SRTP has ability to attain high through put by preserving the header compression techniques efficiently. SRTP, like RTP is suitable for both unicast and multicast environment.

### 1.8.1 Security in SRTP

Conceptually SRTP is considered as bump in the protocol stack implementation, which resides between the application layer and the transport layer. SRTP intercept RTP packet and forwards an equivalent SRTP packet to the receiver, and which incepts SRTP packet and passes an equivalent an RTP packet up the stack to the application layer.

SRTP applies both encryption and authentication to the equivalent RTP packet. The encryption portion contains the payload and padding while authentication portion contains the equivalent RTP packet. For this purpose two optional fields are added to the RTP packet, a MKI field and an authentication tag. MKI is an identifier that is used by key management protocol to identify the master key that derives session keys. The authentication tag is used to carry authentication data. The authentication portion provides protection against replay attacks. A packet is replayed when it is stored by an adversary and then re-injected into the network. SRTP receiver keeps a replay list which keeps record of all the received packets and is authenticated [9].

Secure RTCP follows the definition of Secure RTP. SRTCP adds three mandatory new fields (the SRTCP index, an "encrypt-flag", and the authentication tag) and one optional field (the MKI) to the RTCP packet definition. The three mandatory fields MUST be appended to an RTCP packet in order to form an equivalent SRTCP packet.

The preceding discussion is a brief description of RTP security. The security services like confidentiality, integrity and authentication, key management and any architecture level security service provided will be discussed in subsequent related topics.

## 1.9 JMF

The Java™ Media Framework (JMF) is an Application Programming Interface (API) for incorporating time-based media into Java applications and applets. This guide is intended for Java programmers who want to incorporate time-based media into their applications and for technology providers who are interested in extending JMF and providing JMF plug-ins to support additional media types and perform custom processing and rendering.

### 1.9.1 JMF Architecture

Java™ Media Framework (JMF) provides a unified architecture and messaging protocol for managing the acquisition, processing, and delivery of time-based media data. JMF is designed to support most standard media content types, such as AIFF, AU, AVI, GSM, MIDI, MPEG, QuickTime, RMF, and WAV.

By exploiting the advantages of the Java platform, JMF delivers the promise of "Write Once, Run Anywhere" to developers who want to use media such as audio and video in their Java programs. JMF provides a common cross-platform Java API for accessing underlying media frameworks. JMF implementations can leverage the capabilities of the underlying operating system, while developers can easily create portable Java programs that feature time-based media by writing to the JMF API.

With JMF, you can easily create applets and applications that present, capture, manipulate, and store time-based media. The framework enables advanced developers and technology providers to perform custom processing of raw media data and seamlessly extend JMF to support additional content types and formats, optimize handling of supported formats, and create new presentation mechanisms.

## 1.9.2 High-Level Architecture

Devices such as tape decks and VCRs provide a familiar model for recording, processing, and presenting time-based media. When you play a movie using a VCR, you provide the media stream to the VCR by inserting a video tape. The VCR reads and interprets the data on the tape and sends appropriate signals to your television and speakers.

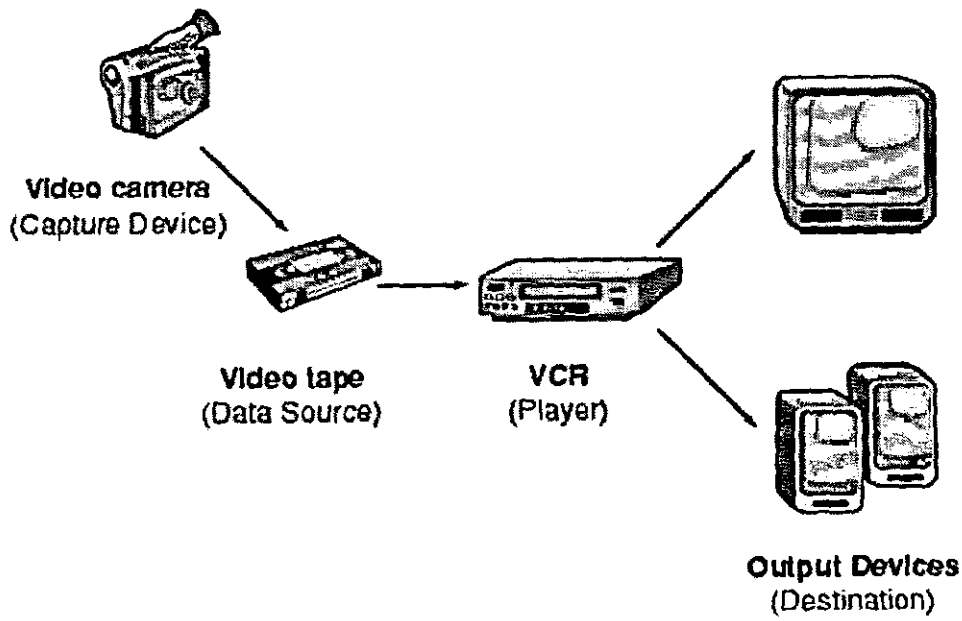


**Figure 1.8: Recording, Processing, and Presenting Time-Based Media**

JMF uses this same basic model. A data source encapsulates the media stream much like a video tape and a player provides processing and control mechanisms similar to a VCR. Playing and capturing audio and video with JMF requires the appropriate input and output devices such as microphones, cameras, speakers, and monitors.

Data sources and players are integral parts of JMF's high-level API for managing the capture, presentation, and processing of time-based media. JMF also provides a lower-level API that supports the seamless integration of custom processing components and extensions. This layering provides Java developers with an easy-to-use API for incorporating time-based media into Java programs while maintaining the flexibility and extensibility required to support advanced media applications and future media technologies [10].

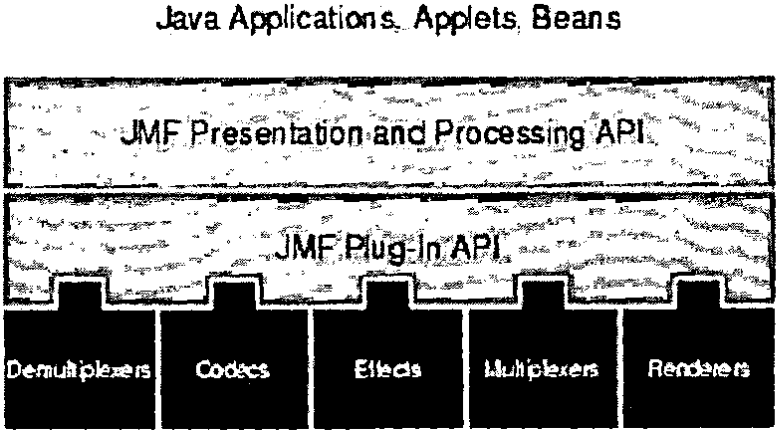


**Figure1.9: High-level JMF Architecture**

## 1.9.3 Understanding the JMF-RTP API

JMF enables the playback and transmission of RTP streams through the APIs defined in the following packages:

1. javax.media.rtp
2. javax.media.rtp.event
3. javax.media.rtp.rtcp

JMF can be extended to support additional RTP-specific formats and dynamic payloads through the standard JMF plug-in mechanism.

**Note:** JMF-compliant implementations are not required to support the RTP APIs in javax.media.rtp, javax.media.rtp.event, and javax.media.rtp.rtcp. The reference implementations of JMF provided by Sun Microsystems, Inc. and IBM Corporation fully support these APIs.

You can play incoming RTP streams locally, save them to a file, or both.

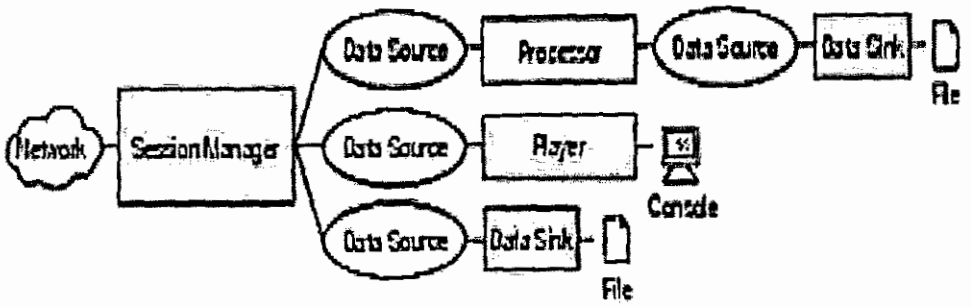


**Figure 1.10: RTP Reception**

For example, the RTP APIs could be used to implement a telephony application that answers calls and records messages like an answering machine.

Similarly, you can use the RTP APIs to transmit captured or stored media streams across the network. Outgoing RTP streams can originate from a file or a capture device. The outgoing streams can also be played locally, saved to a file, or both.

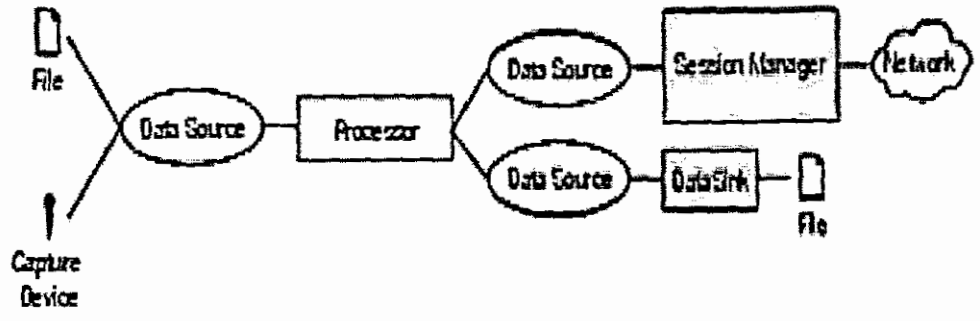


**Figure 1.11: RTP Transmission**

For example, you could implement a video conferencing application that captures live audio and video and transmits it across the network using a separate RTP session for each media type.

T-1068

Similarly, you might record a conference for later broadcast or use a prerecorded audio stream as "hold music" in a conferencing application.

## 1.10 Working with Real-Time Media Streams

To send or receive a live media broadcast or conduct a video conference over the Internet or Intranet, you need to be able to receive and transmit media streams in real-time. This chapter introduces streaming media concepts and describes the Real-time Transport Protocol JMF uses for receiving and transmitting media streams across the network.

### 1.10.1 Streaming Media

When media content is streamed to a client in real-time, the client can begin to play the stream without having to wait for the complete stream to download. In fact, the stream might not even have a predefined duration, downloading the entire stream before playing it would be impossible. The term streaming media is often used to refer to both this technique of delivering content over the network in real-time and the real-time media content that's delivered.

Streaming media is everywhere you look on the web, live radio and television broadcasts and webcast concerts and events are being offered by a rapidly growing number of web portals, and it's now possible to conduct audio and video conferences over the Internet. By enabling the delivery of dynamic, interactive media content across the network, streaming media is changing the way people communicate and access information.

### 1.10.2 Protocols for Streaming Media

Transmitting media data across the net in real-time requires high network throughput. It's easier to compensate for lost data than to compensate for large delays in receiving the

data. This is very different from accessing static data such as a file, where the most important thing is that all of the data arrive at its destination. Consequently, the protocols used for static data don't work well for streaming media.

The HTTP and FTP protocols are based on the Transmission Control Protocol (TCP). TCP is a transport-layer protocol[1] designed for reliable data communications on low-bandwidth, high-error-rate networks. When a packet is lost or corrupted, it's retransmitted. The overhead of guaranteeing reliable data transfer slows the overall transmission rate.

For this reason, underlying protocols other than TCP are typically used for streaming media. One that's commonly used is the User Datagram Protocol (UDP). UDP is an unreliable protocol; it does not guarantee that each packet will reach its destination. There's also no guarantee that the packets will arrive in the order that they were sent. The receiver has to be able to compensate for lost data, duplicate packets, and packets that arrive out of order.

Like TCP, UDP is a general transport-layer protocol, a lower-level networking protocol on top of which more application-specific protocols are built. The internet standard for transporting real-time data such as audio and video is the Real-Time Transport Protocol (RTP).

### 1.10.3 Transmitting Media Streams across the Network

RTP server applications transmit captured or stored media streams across the network.

For example, in a conferencing application, a media stream might be captured from a video camera and sent out on one or more RTP sessions. The media streams might be encoded in multiple media formats and sent out on several RTP sessions for conferencing with heterogeneous receivers. Multiparty conferencing could be implemented without IP multicast by using multiple unicast RTP sessions.

### 1.10.4 Receiving Media Streams from the Network

Being able to receive RTP streams is necessary for several types of applications. For example:

1. Conferencing applications need to be able to receive a media stream from an RTP session and render it on the console.
2. A telephone answering machine application needs to be able to receive a media stream from an RTP session and store it in a file.
3. An application that records a conversation or conference must be able to receive a media stream from an RTP session and both render it on the console and store it in a file.

# CHAPTER 2

# Problem Domain
# &
# System Analysis

## 2. Existing System

In the existing real time applications security has been a main issue. The RFC recommended algorithms (DES, 3DES & AES) have not been implemented as yet in any application altogether. The existing applications either focused on enhancing voice or video quality or their data rate (Transmission Speed).

## 2.1 Proposed System

In the new proposed system a user will be able to send encrypted real time data over the underlying network that uses UDP protocol with multicast capabilities. The sender can select an algorithm from a list of RFC recommended security algorithms to send data. Only the receiver will be able to decrypt the incoming RTP stream since he knows the encryption algorithm that has been applied. The new system should be able to work on any operating system.

### 2.1.1 Project Definition

In the secure video conferencing system TranSecure many users can participate at the same time. They can send and receive data at the same time. Only one person at a time will be able to speak and all others will have to listen until the active user is deactivated.

The speaking/active person will be able to watch all other participants while his address. So that he could notice the facial expressions and moments in reaction of his speech from the participant's side. Proposed software will overcome the drawback of existing systems as it can become platform independent and faster depending upon the LAN specifications.

### 2.1.2 Project Scope

The new application can be used by any organization for a secure real time communication session. The end users of TranSecure can be

1. Hospitals for Telemedicine sessions
2. Military departments
3. Universities for delivering distant education
4. Businessmen for communication with clients
5. Oil and gas companies to have a close look on their sites
6. News agencies for delivering live info from field
7. Shopping malls for keeping an eye on shoppers
8. From remote location for interviews of applicants and many more

Any organization can use the software for the purposes stated above. So it can be easily commercialized. Many organizations are looking to transfer their old manual conferencing system to the latest video conferencing systems. These days' markets are rapidly getting developed in this new field.

### 2.1.3 Efficiency

Due to the usage of latest CODECS data will travel on LAN in an efficient way. Although the data rate will be a little slow when a security algorithm is applied but alternately you have to pay something for the a secure RTP session.

### 2.1.4 Reliability

As the application is developed using state of the art technology and algorithms it will be highly reliable for a lot of reasons the foremost being the security aspect. The sender can be sure that his data has been sent to the receiver after receiving the receiver report (RR). He can check the transmission status by checking the receiver report. As the development environment is JAVA the users can be sure of a secure RTP session.

### 2.1.5 User Friendliness

The interface is made in such a way to provide the user-friendly environment. Even a lay man would be able to use the software easily. The interfaces are very easily built up keeping in mind all the possibilities and according to the proposed approaches of software engineering.

### 2.1.6 Future Enhancements

The software can be implemented interdepartmentally by performing nominal modifications. The reason for this easy conversion is that it will use TCP/IP protocol suite and object oriented programming in JAVA. So the small modules can be reused to build up a more complex and valuable software.

### 2.1.7 Advantages of the Proposed System

Proposed system has the following advantages:

1. Multiple users can use this application at the same time.
2. No eavesdropper can decrypt the RTP stream if it is encrypted.
3. Its works on any type of networks.

4. This application emulates the real time conference.
5. Every one can use the application with great ease.
6. Any user can participate in the conference even if he does not have the audio or video equipment.
7. The application can be commercialized easily.
8. The application has the ability to be enhanced and new features like file sharing can be easily added.
9. Application uses very less bandwidth.
10. The application uses the latest CODECS for audio and video coding and decoding.
11. No extra rearrangements are required for the application to work on LAN. It works on the existing scenario.

## 2.2 System Analysis

In this process the problem is analyzed to see that how it can meet requirements. The system is divided into different components and their functionality is decided. The interaction of the system components with one another and the actors are analyzed.

### 2.2.1 Object

Objects are the entities of our system. In design phase, these will be the classes of our system. All nouns in our problem statements will be candidates for objects. There is a criterion for selecting the objects. The candidates that fulfill the following requirements will be the objects of our system.

Following are the candidate objects

1. Window media player
2. Sender
3. Receiver
4. Client
5. Encryption Algorithms
6. Interface
7. Local Area Network (LAN)
8. Terminals
9. Camera
10. Headphone
11. Microphone
12. Real Time Transport Protocol

### 2.2.2 Criteria for Objects

How do we know whether a potential object is a good candidate for use in an Object Oriented system? The following is the criteria for selecting an object.

#### 2.2.2.1 Retained Information

The specific object will be useful during analysis only if information about it must be remembered so that system can function.

### 2.2.2.2 Needed Services

The specific object must have a set of identifiable operations that can change the value of its attributes in some way.

### 2.2.2.3 Multiple Attribute

During requirement analysis focus should be on important information i.e.; a single attribute may infect be useful during design but it is probably better represented as an attribute of another object during the analysis activity.

### 2.2.2.4 Common Attributes

A set of attributes can be defined for the specific objects and these attributes applied to all occurrences of objects.

### 2.2.2.5 Common Operations

A set of operations can be defined for the specific objects and these operations applied to all occurrences of objects.

### 2.2.2.6 Essential Requirements

External entities that appear in the problem space and produce or consume information that is essential to the operation of any solution for the system will almost always be defined as objects in the requirement modal.

### 2.2.3 Selected Objects

Following objects were fulfilling the above stated selection criteria for objects. These objects will become our classes.

1. Sender
2. Receiver
3. Client
4. Encryption Algorithms
5. Interface
6. Local Area Network (LAN)
7. Terminals
8. Camera
9. Headphone
10. Microphone
11. Real Time Transport Protocol

## 2.3 Domain Analysis

Domain analysis is an important step during the analysis phase. In technical terms it can be defined as, "domain analysis is the identification analysis and specification of common requirements from a specification of common requirements from a specific application domain".

In simple words it is "The identification, analysis and specification of common reusable capabilities with in a specific application domain, in terms of common objects, classes, sub assemblies and frame work".

There are following steps in domain analysis.

1. Domain to be investigated
2. Categorization of objects

Now implementation of these steps will be as follows.

### 2.3.1 Domain to be Investigated

During this step there are two subcategories.

#### 2.3.1.1 Interested Domain

Our interested domain is to provide opportunity to users, sitting apart from each other on their own computer systems connected to the same LAN, to interact with each other as in real time conference. Users may also send text messages and documents to one another.

#### 2.3.1.2 Object Oriented Items

Object oriented items include specifications, code for existing object oriented application classes. Our object oriented items are as follows.

**Media Classes**

1. javax.media.*
2. javax.media.rtp.*
3. javax.media.rtp.event.*

4. javax.media.rtp.rtcp.*
5. javax.media.protocol.*
6. javax.media.protocol.DataSource
7. javax.media.format.AudioFormat
8. javax.media.format.VideoFormat
9. javax.media.Format
10. javax.media.format.FormatChangeEvent
11. javax.media.control.BufferControl

These packages provide the classes and interfaces to send, receive and control the media related features.

## Networking Classes

1. java.net.*
2. java.io.*

These packages provide the classes and interfaces to provide communication between the server and the clients.

## Encryption Classes

1. javax.crypto.*
2. javax.cypto.spec.*
3. javax.security.*

These packages provide the encryption algorithms to secure the payload transmitted on the underlying protocol.

### Interface Classes

1. java.awt.*
2. javax.swing.*
3. javax.swing.border.*

These packages provide the classes and interfaces to create the application to human interfaces.

### 2.3.1.3 Non Object Oriented Items

Non object oriented items include the hardware and software. Following items are used as non object oriented item in our project.

1. Microsoft Word
2. Microsoft Paint
3. Web Cam

# CHAPTER 3

# SYSTEM DESIGN

# 3. System Design

In design mode we give shape to our components for implementation. Different classes are created. These classes represent the functions and attributes to be used in implementation. The objects in object oriented analysis will be the classes for our system. In structure analysis these are the entities of our system. We shell discuss these classes and their functionality in detail here. The class diagram will show the variable and operation of that class and relationship among different classes

## 3.1 Important Classes

Following classes are used in TranSecure

1. Main
2. Config
3. Transmitter
4. Receiver
5. Server
6. Client

### Main

This class creates the graphic user interface of the application. It adds and removes a target from the receiver or senders list. It also creates socket.

### Config

This class stores the local and remote addresses in a log file so that the data can be reloaded once the application is restarted.

## Transmitter

Transmitter is a main class of our system. Through this very class we are able to transmit the video over the LAN. Its basic functionality is to capture the video from the video source and compels it to be transmitted over the LAN. Here in this class first of all we take the list of all the attached devices from the system through the built in function of the Java's API JMF. Then we set the format for the transmission e.g. MPEG, AVI and many more formats are there. Thirdly we create a RTP SESSION MANAGER which is very important step in our class. This manager provides us the facility to transmit the data from a real time capture source like in this case is camera. Through this process we transmit the video of a user.

## Receiver

This is another very important class of our system. Here the receiver class receives a stream from its interface named LISTENER. Then it creates a data source of the received stream being received by the transmitter class of the transmitting system. This class also needs to make a RTP SESSION MANAGER, because it has to work with the real time media. Here this class modifies the stream by specifying the CODECS for the video and settles down the related issues. Then this class performs a last step and that is to create a video player from data source, to make the video visible to the receiving user.

## Server

This class creates a connection with the client, and instructs the client class to use a specified security algorithm that has been used at the transmitting end. This class also encrypts the data using the security algorithm selected by the user.

### Client

This class receives the data from the server class. It decrypts the data and passes the decrypted data to the application for display.

## 3.2 Use Cases

Use cases are the tasks that are performed by a user of the system. The user, known as an actor in UML invokes use cases. In coming sections we will discus the use cases of our system and also extended use case formats.

### 3.2.1 Actors of the System

An Actor is any thing that communicates with the system or the product that is external to the system usually an actor represents the roles that people are devices play as the system operator.

The following are the actors of our system:

### 1. Sender

Sender will initiate the RTP session. He will enter the address of the receiver and will start transmission. If the sender wants to encrypt data he can do so by selecting a security algorithm from the application.

### 2. Receiver

Receiver will receive the data from sender. If the data is encrypted he can decrypt it.

### 3.2.2 Domains in our System.

There are two domains in our system.

1. Transmitter
2. Receiver

Therefore the use cases will be divided into two sub domains.

### 3.2.3 Use cases of Transmitter

1. Add Target
2. Remove Target
3. Select Cipher
4. Select Data Source
5. Start Transmission
6. Stop Transmission
7. Maintain Log File

### Add Target

The transmitter will add the IP address and the Data Port of the receiver to add a target. It can add multiple targets if wanted. Every receiver will have its own session with the sender.

## Remove Target

If the transmitter wants to stop a receiver from receiving RTP stream it can do so by removing the receiver from the target list.

## Select Cipher

If the transmitter wants to send encrypted data it can do it by selecting a cipher type from the menu. The three available ciphers are

1. Data Encryption Standard (DES)
2. Triple Data Encryption Standard (Triple DES)
3. Advanced Encryption Standard (AES)

The default option is Null cipher that leaves data unencrypted.

## Select Data Source

Transmitter can select the data source. If it wants to send live data from the camera it can do so by selecting the video camera as the data source. If it wants to send data from a file it can also do this by giving the location of the media file.

## Start Transmission

Transmitter can start sending data by starting the transmission process. After the transmission has been started all the receivers will start receiving data.

## Stop Transmission

If the transmitter wants to stop the data transmission it can do so by stopping the transmission process. After the transmission has been stopped no one will be receiving data anymore.

## Maintain Log File

A log of target client is stored in a file "Xmit.dat" that reloads the list of receivers in the application when restarted.

### 3.2.4 Use Cases of Receiver

1. Add Target
2. Remove Target
3. Start Receiver
4. Decrypt Data
5. Stop Receiver
6. Maintain Log File

## Add Target

The receiver will add the IP address and the Data Port of the sender to add a target. It can add multiple targets if he wants. Every sender will have its own session with receiver for which a separate player window will open.

## Remove Target:

If the receiver wants to stop receiving data from the sender it can do it by removing the sender from the target list.

## Start Receiver

Receiver can start receiving data by starting the reception process. It can receive from multiple targets if wanted. Every sender will have its own session with the receiver.

## Decrypt Data

If the transmitter wants to send encrypted data it can do it by selecting a cipher type from the available ciphers i.e.

1. Data Encryption Standard (DES)
2. Triple Data Encryption Standard (Triple DES)
3. Advanced Encryption Standard (AES)

The receiver will have a separate socket opened that will provide the name of cipher that is encrypting data and accordingly will decrypt data.

## Stop Receiver

If the receiver wants to stop receiving data it can do so by stopping the transmission process. After the transmission has been stopped it will be receiving no data anymore.

### Maintain Log File

A list of senders is stored in a file "rx.dat" that reloads the list of senders in the application when restarted.

## 3.2.5 Use Cases in Extended Format

In extended use case we have to provide the detail about the use case.

## 3.2.6 Extended Use Cases of Transmitter

### 1. Add target

Name: Add Target

Actor: User

Pre condition: No Precondition

Post condition: Add receiver to log file

Description: The transmitter will add the IP address and the Data Port of the receiver to add a target. It can add multiple targets if wanted. Every receiver will have its own session with the sender.

Type: Primary, Essential.

| Actor Action | System Response |
|---|---|
| 1. The user adds IP address, remote data port and local data port | 3. System adds target to list. If the address already exists, system would not add target |
| 2. The user clicks add target button. | 4. System adds target to log file. |

## 2. Remove Target

Name: Remove Target

Actors: User

Pre condition: Target must be in the log file

Post condition: Target is removed from the log file

Description: If the transmitter wants to stop a receiver from receiving RTP stream it can do so by removing the receiver from the target list.

Type: Primary, Essential.

| Actor Action | System Response |
|---|---|
| 1. User clicks the address of a receiver in the receivers list. | 3. System removes the target from the list. |
| 2. User clicks the remove target button. | 4. System removes target from the log file. |

## 3. Select Cipher

Name: Select Cipher

Actor: User

Pre condition: Some data must be captured.

Post condition: Data has to be transmitted with encryption.

Description: If the transmitter wants to send encrypted data it can do so by selecting a cipher type from the menu. The three available ciphers are

1. Data Encryption Standard (DES)
2. Triple Data Encryption Standard (Triple DES)

3. Advanced Encryption Standard (AES)

The default option is Null cipher that leaves data unencrypted.

Type: Primary, Essential.

| Actor Action | System Response |
|---|---|
| 1. User will select a cipher type from the menu. | 2. The data will be encrypted using the selected cipher. |

**Alternate Course**

| | |
|---|---|
| 1. By default NULL cipher is used for encryption. | 2. Data will be unencrypted if transmitted. |

## 4. Select Data Source

Name: Select data Source

Actors: User

Pre condition:

1. File must exist
2. Data capture device (Camera) must exist.

Post condition: Application is capturing data successfully.

Description: Transmitter can select the data source. If it wants to send live data from the camera it can do so by selecting the video camera as the data source. If it wants to send data from a file it can also do this by giving the location of the media file.

Type: Primary, Essential

| Actor Action | System response |
|---|---|
| 1. User will select data source. | 2. Application will start capturing data from the selected data source. |

**Alternate Course**

| | |
|---|---|
| 1. If no data source is selected. | 2. Application will generate an error "Data source not found". |

## 5. Start Transmission

Name: Start Transmission

Actors: User

Pre condition: data source and target must have been added.

Post condition: data will be sent to the receiver.

Description: In this use case transmitter can start sending data by starting the transmission process. After the transmission has been started all the receivers will start receiving data.

Type: Primary, Essential

| Actor Action | System response |
|---|---|
| 1. Transmitter will press the start transmission button. | 2. Data will be transmitted to the receiver. |

## 6. Stop Transmission

Name: Stop Transmission

Actors: User

Pre condition: Transmission has been started.

Post condition: Transmission will be stopped.

Description: In this use case if the transmitter wants to stop the data transmission it can do so by stopping the transmission process. After the transmission has been stopped no one will be receiving data anymore.

Type: Primary, Essential

| Actor Action | System response |
|---|---|
| 1. Transmitter will press the stop transmission button. | 2. Data transmission to the receiver will stop. |

## 7. Maintain Log File

Name: Maintain log file

Actors: Transmitter

Pre condition: Any target has been added to the receivers list.

Post condition: Targets address will be added to the log file.

Description: In this use case a log of target clients is stored in a file "Xmit.dat" that reloads the list of receivers in the application when restarted.

Type: Primary, Essential

| Actor Action | System response |
|---|---|
| 1. User will click the add target button. | 2. Receiver data will be added to the log file. |

### 3.2.7 Extended Use Cases of Receiver

### 1. Add target

Name: Add target

Actor: User

Pre condition: No Precondition

Post condition: Add sender to log file

Description: The receiver will add the IP address and the Data Port of the sender to add a target. It can add multiple targets if wanted. Every sender will have its own session with the receiver.

Type: Primary, Essential.

| Actor Action | System Response |
|---|---|
| 1. The user adds IP address, remote data port and local data port | 3. System adds target to list. If the address already exists, system would not add target |
| 2. The user clicks add target button. | 4. System adds target to log file. |

### 2. Remove Target

Name: Remove Target

Actors: User

Pre condition: Target must be in the log file

Post condition: Target is removed from the log file

Description: If the receiver wants to stop a sender from sending RTP stream it can do so by removing the sender from the target list.

Type: Primary, Essential.

| Actor Action | System response |
|---|---|
| 1. User clicks the address of a sender in the senders list. | 3. System removes the sender from the list. |
| 2. User clicks the remove target button. | 4. System removes sender from the log file. |

### 3. Start Receiver

Name: Start Receiver

Actors: User

Pre condition: Data source and target must have been added.

Post condition: Data will be received from sender.

Description: In this use case receiver can start receiving data by starting the transmission process. After the transmission has been started all the senders in the senders list will start sending data to the receiver.

Type: Primary, Essential

| Actor Action | System response |
|---|---|
| 1. Receiver will press the start receiver button. | 2. Data will be received by the receiver. |

## 4. Decrypt Data

Name: Decrypt Data

Actor: Receiver

Pre condition: Data sent by sender is encrypted.

Post condition: Data is decrypted and read by the player.

Description: If the transmitter wants to send encrypted data it can do it by selecting a cipher type from the available ciphers i.e.

1. Data Encryption Standard (DES)
2. Triple Data Encryption Standard (Triple DES)
3. Advanced Encryption Standard (AES)

The receiver will have a separate socket opened that will provide the name of cipher that is encrypting data and accordingly will decrypt data.

Type: Primary, Essential.

| Actor Action | System response |
|---|---|
| 1. User will click the Decrypt Data button. | 2. The data will be Decrypted using the cipher whose name will be provided by the sender. |

### Alternate Course

| | |
|---|---|
| 1. By default NULL cipher is used for encryption. | 2. Data will not be decrypted if transmitted. |

## 5. Stop Receiver

Name: Stop Receiver

Actors: User

Pre condition: Receiver is receiving data.

Post condition: Data reception will be stopped.

Description: In this use case if the receiver wants to stop receiving the data, it can do so by stopping the reception process. After the receiver has been stopped no one will be receiving data anymore.

Type: Primary, Essential

| Actor Action | System response |
|---|---|
| 1. User will press the stop receiver button. | 2. Data transmission to the receiver will stop. |

## 6. Maintain Log File

Name: Maintain log file

Actors: Receiver

Pre condition: Any target has been added to the senders list.

Post condition: Target address will be added to the log file.

Description: In this use case a log of target clients is stored in a file "rx.dat" that reloads the list of senders in the application when restarted.

Type: Primary, Essential

| Actor Action | System response |
|---|---|
| 1. User will click the add target button. | 2. Senders data will be added to the log file. |

## 3.3 Use-Case Diagrams

User diagram is the schematic representation of the Use-Case and their interaction with the actors. Use-Case diagrams for our project are as follows.

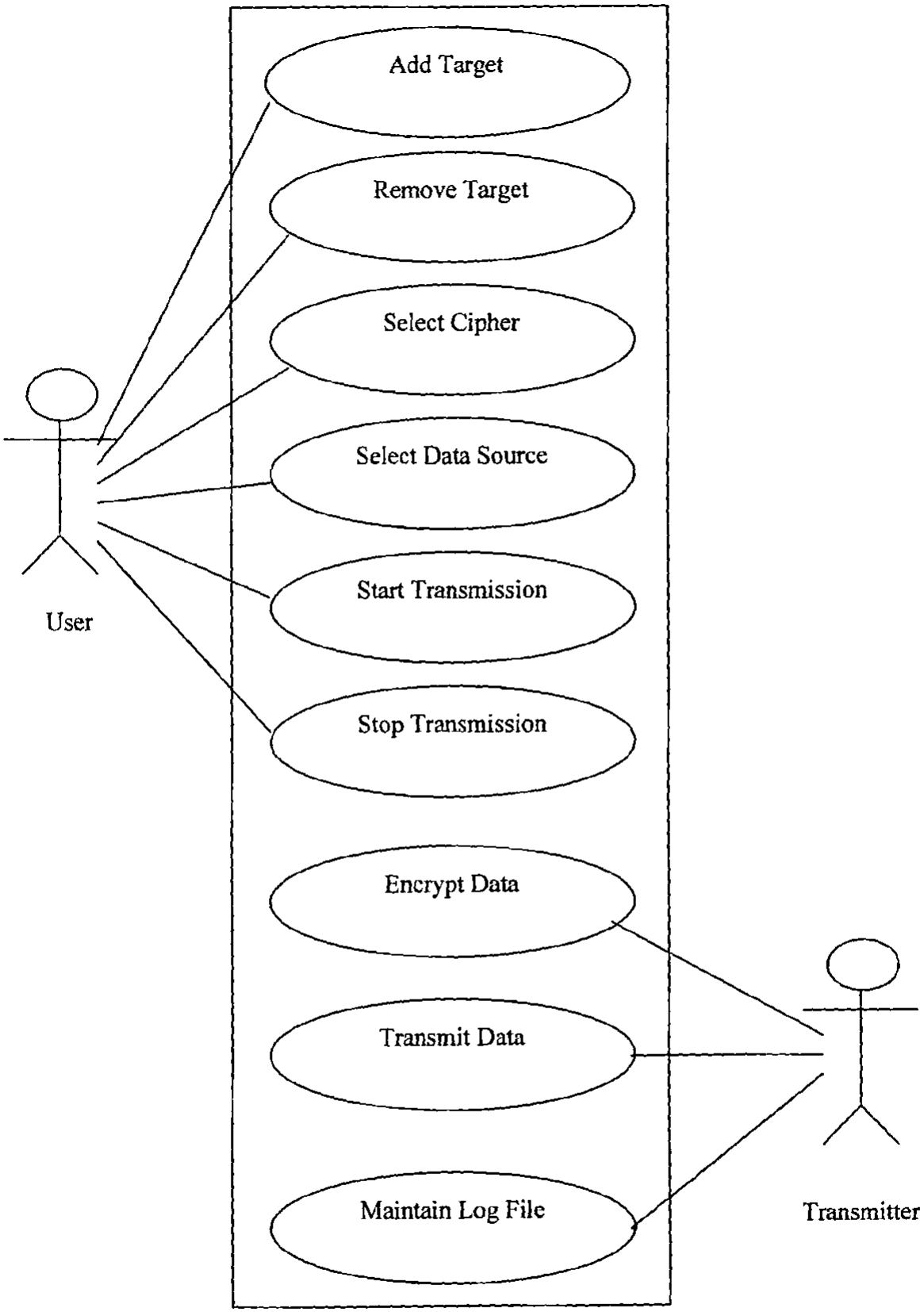


**Fig 3.1: Use Case Diagram of Transmitter**

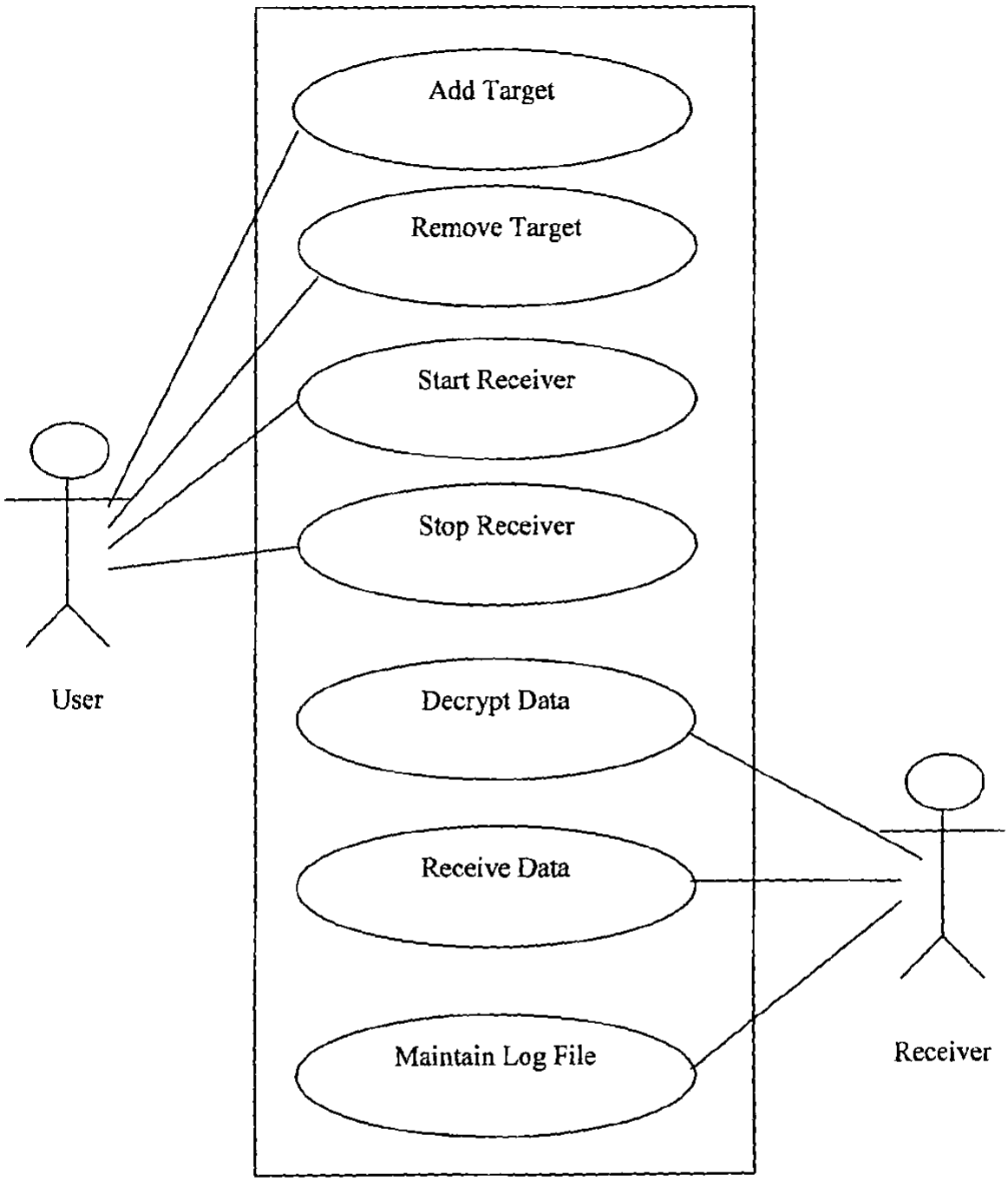


**Fig 3.2: Use Case Diagram of Receiver**

## 3.4 Sequence diagrams

Sequence diagram is the diagram or design artifact that represents the sequence of activities. The following diagrams represent the sequence of activities in TranSecure.

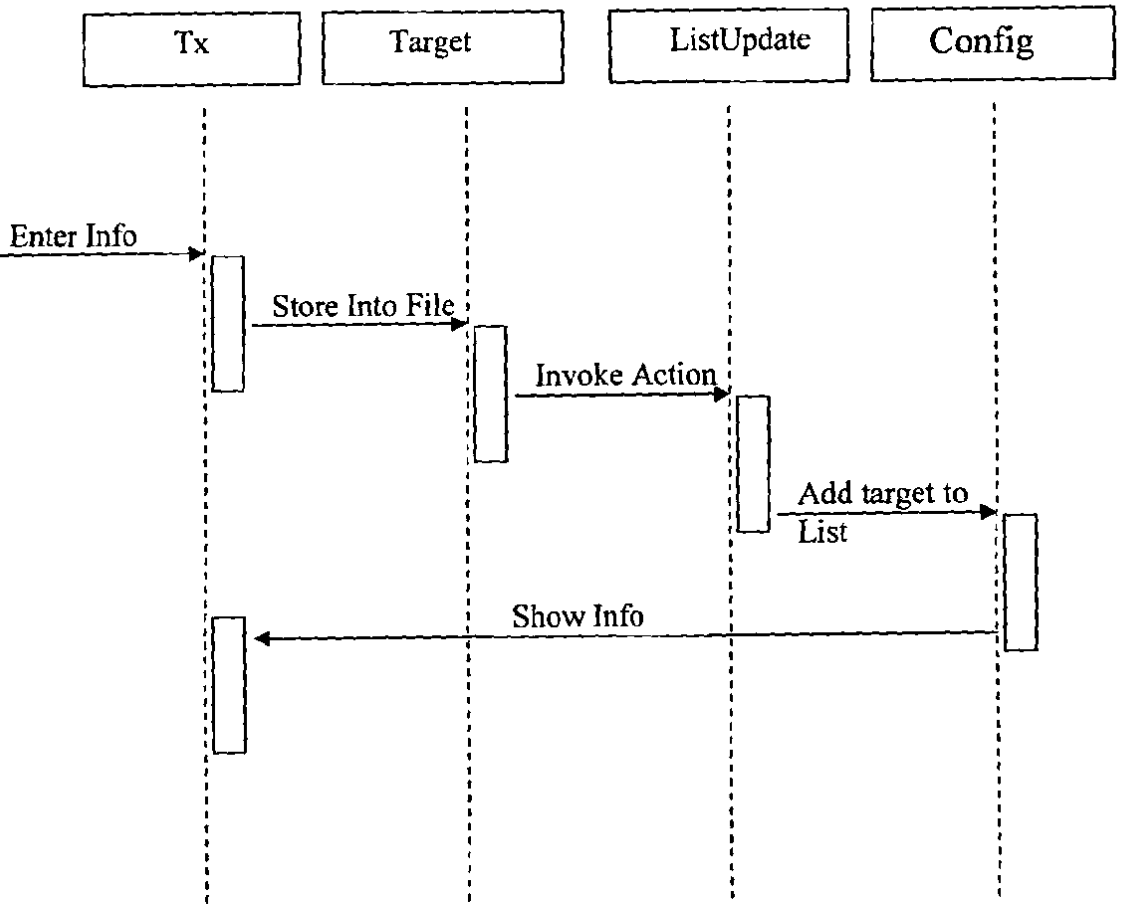


**Fig 3.3: Sequence diagram for Add Target Use-Case**

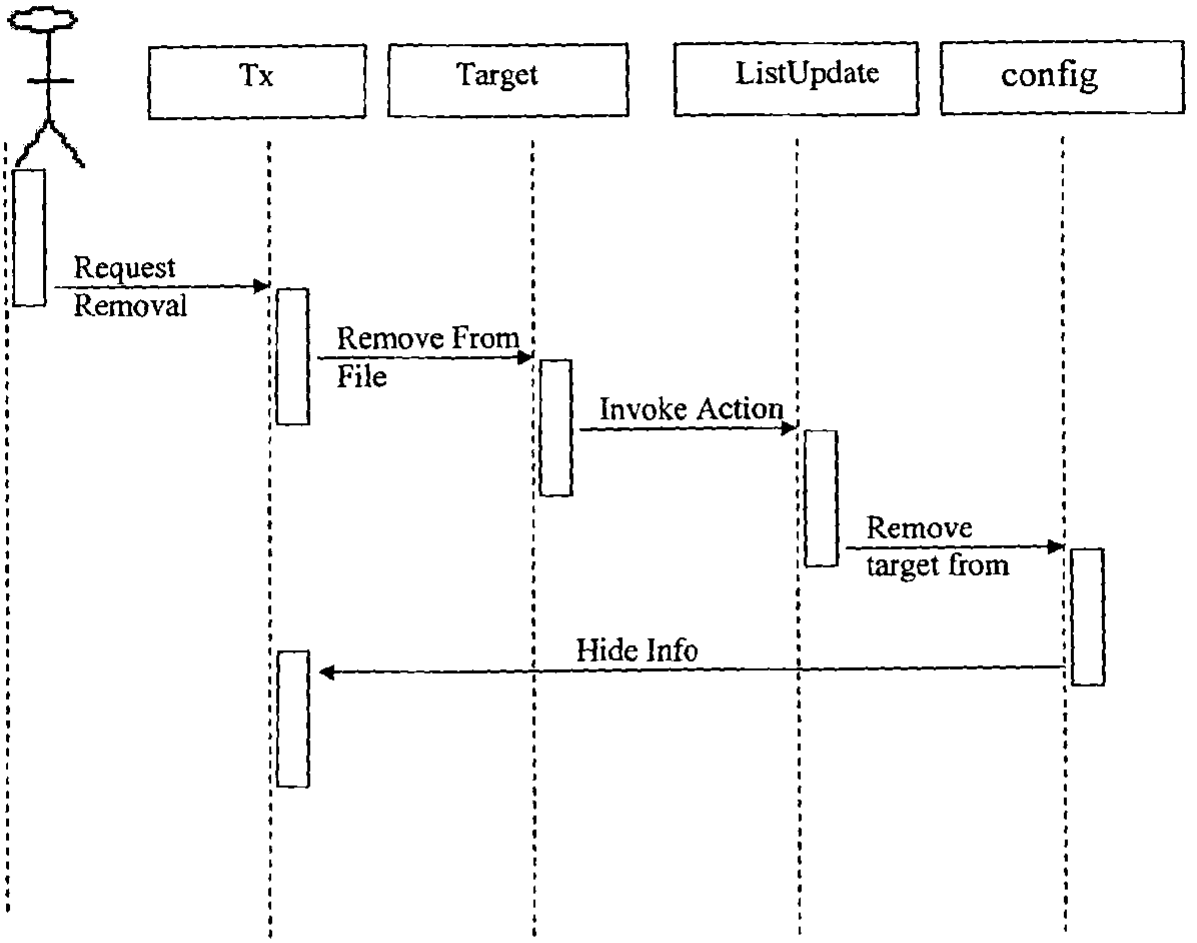


**Fig 3.4: Sequence diagram for Remove Target Use-Case**

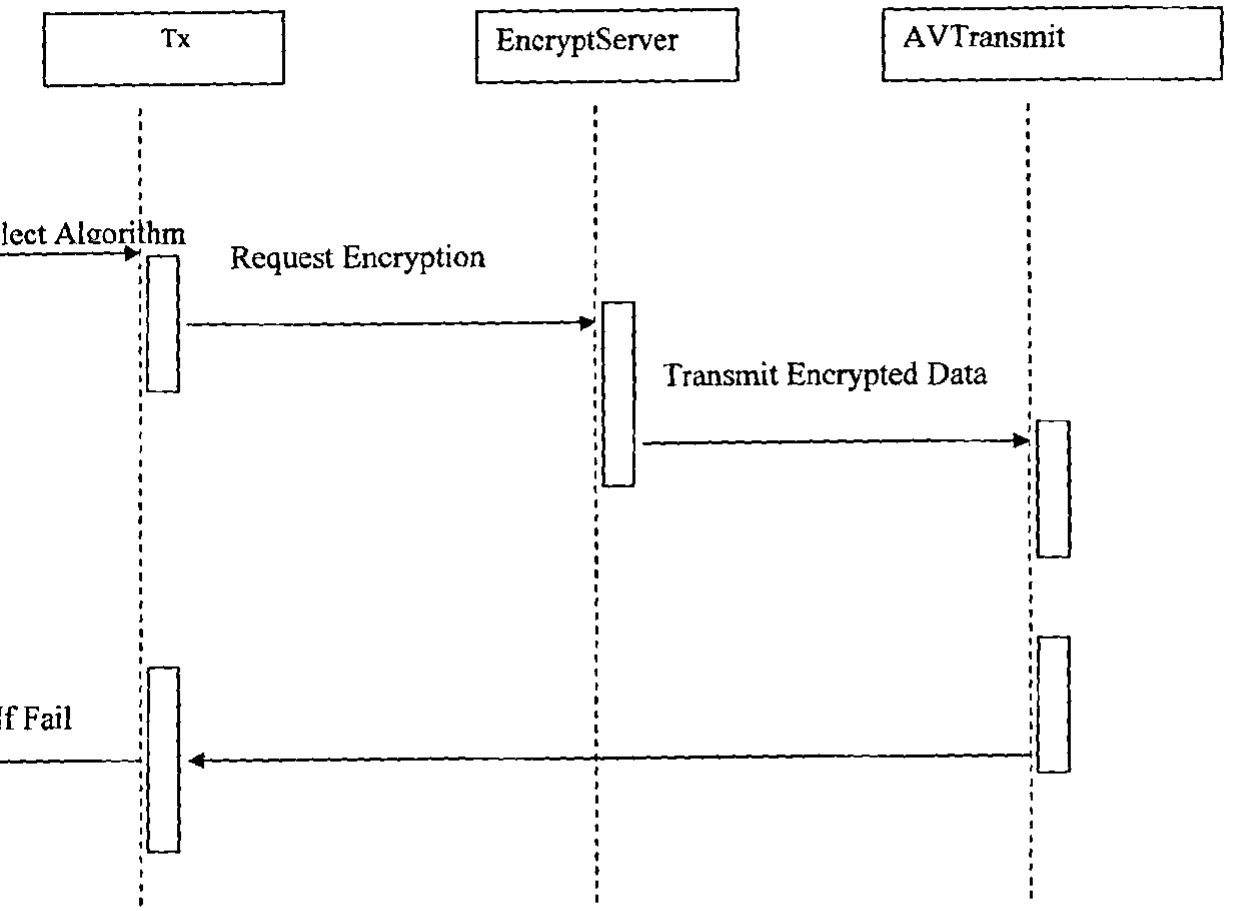


**Fig 3.5: Sequence diagram for Encrypt Data Use-Case**

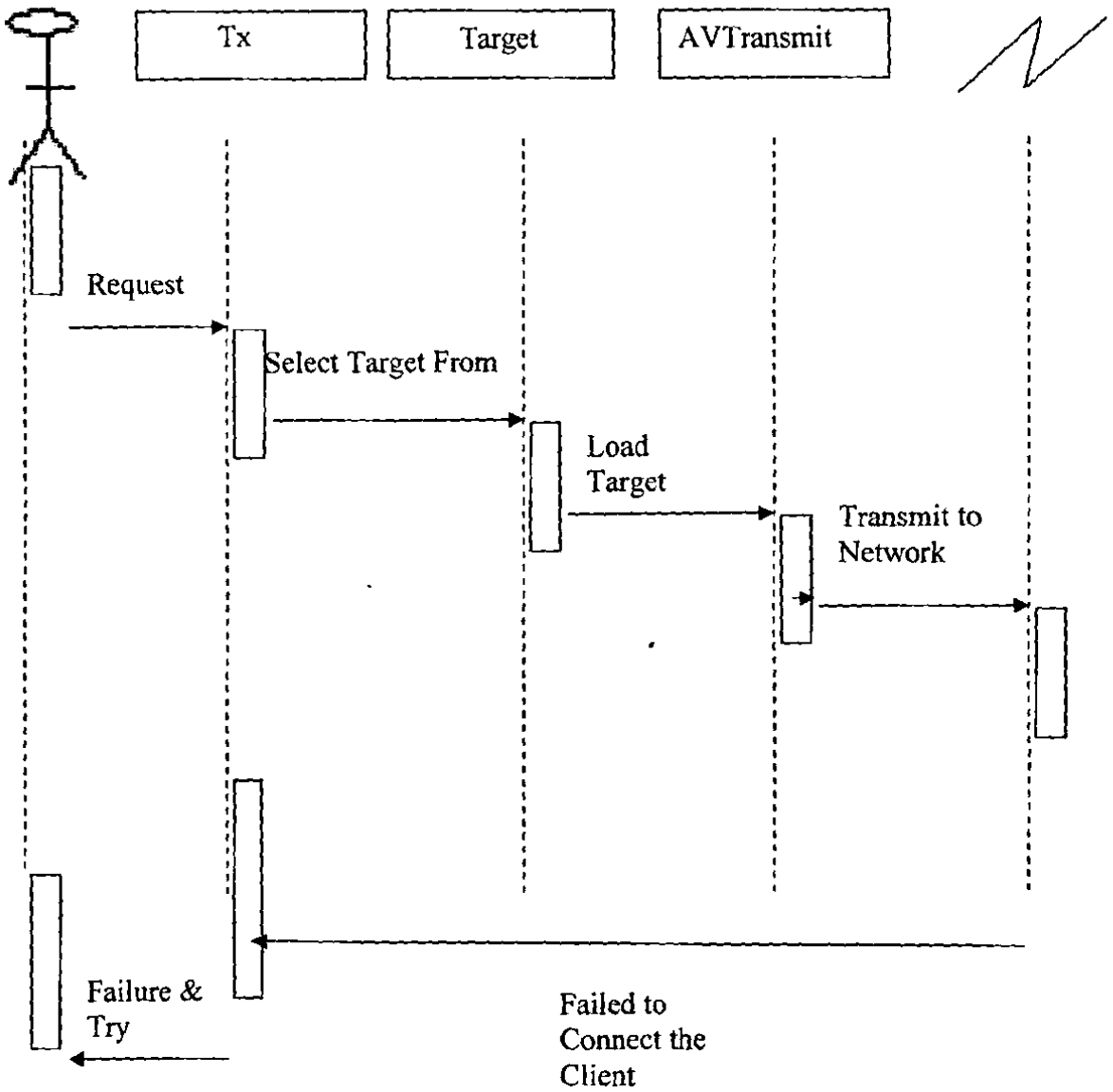


**Fig 3.6: Sequence diagram for Start Transmission Use-Case**

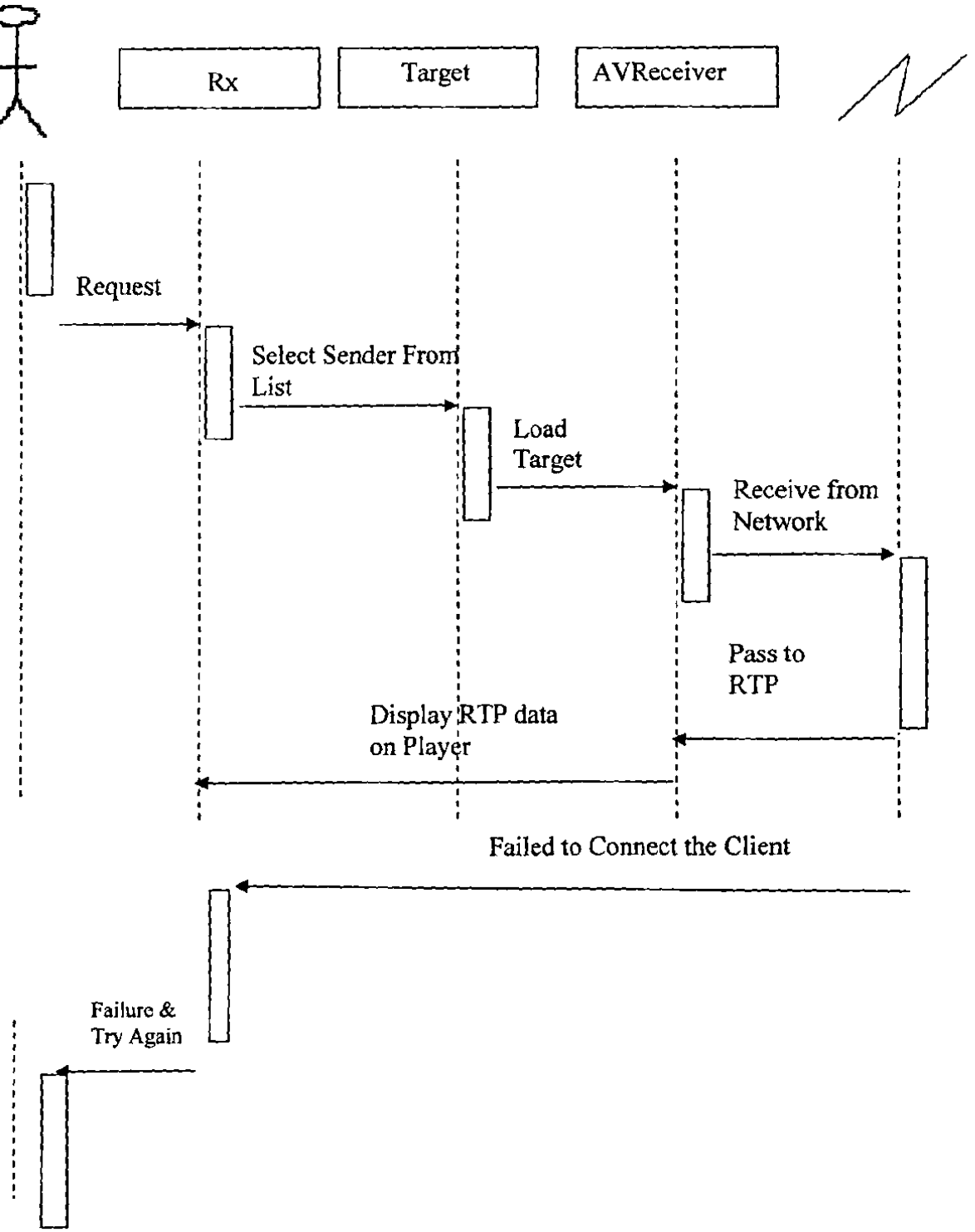


**Fig 3.7: Sequence diagram for Start Receiver Use-Case**

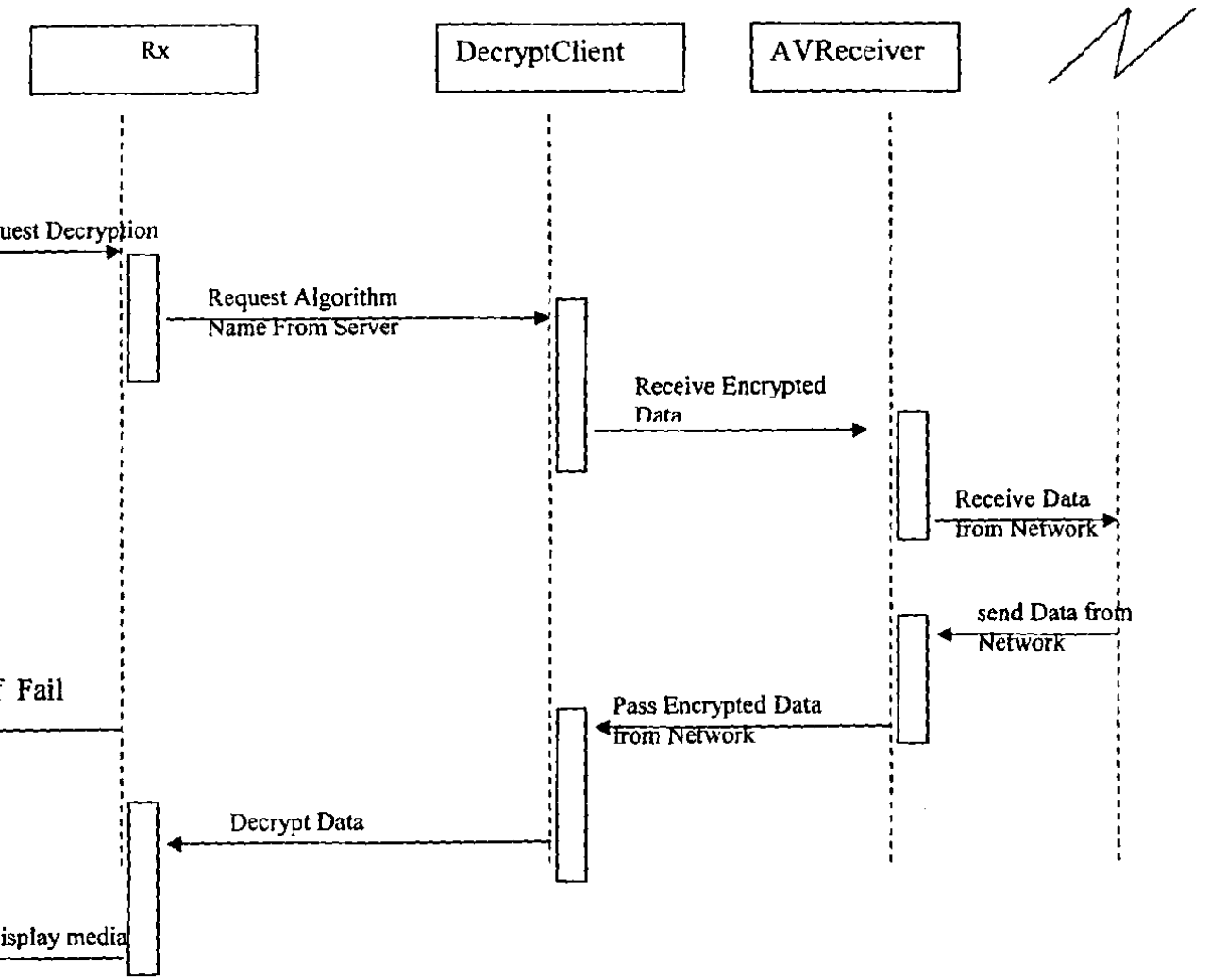


**Fig 3.8: Sequence diagram for Decrypt Data Use-Case**

## 3.5 Object Diagrams

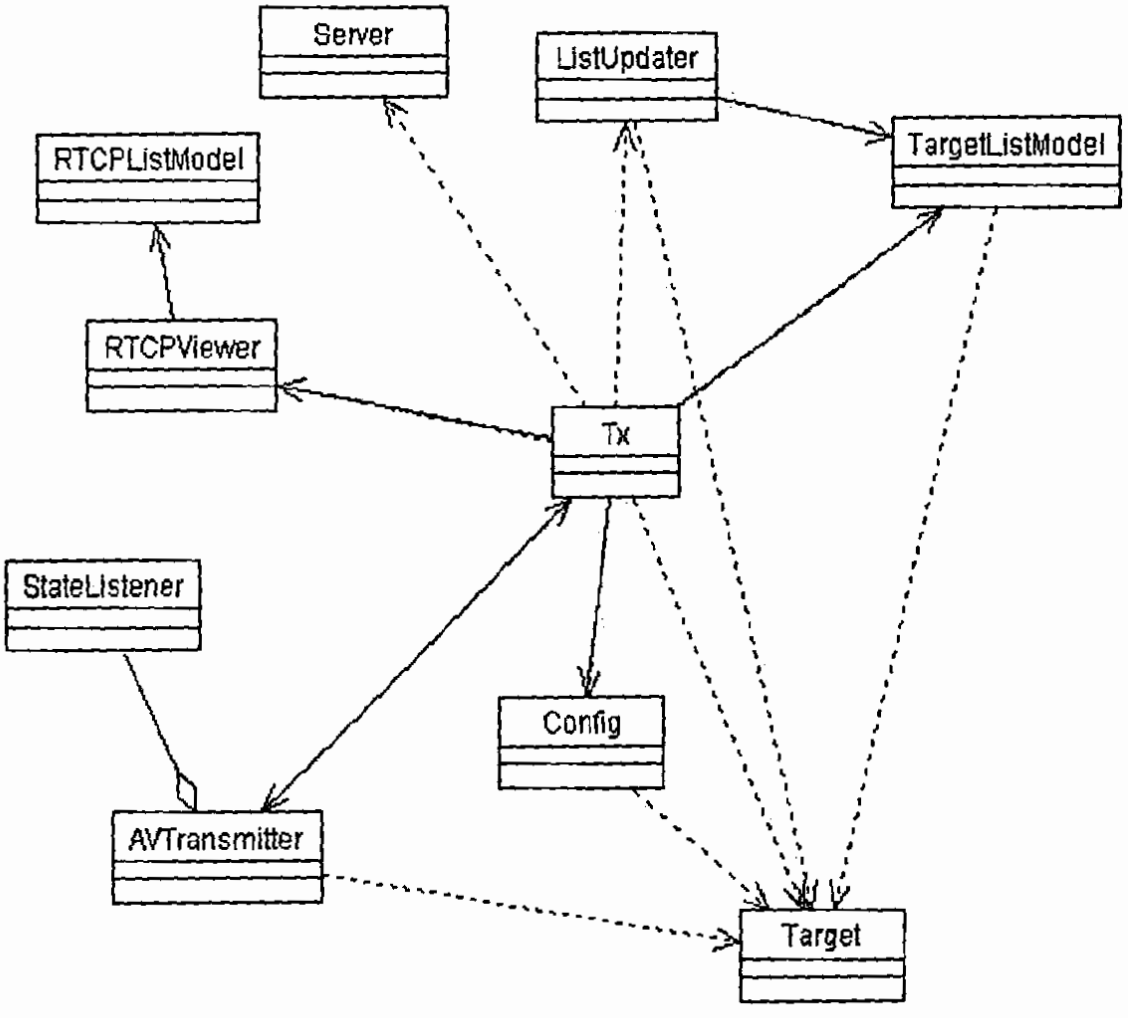


**Fig 3.9: Object diagram of Transmitter**

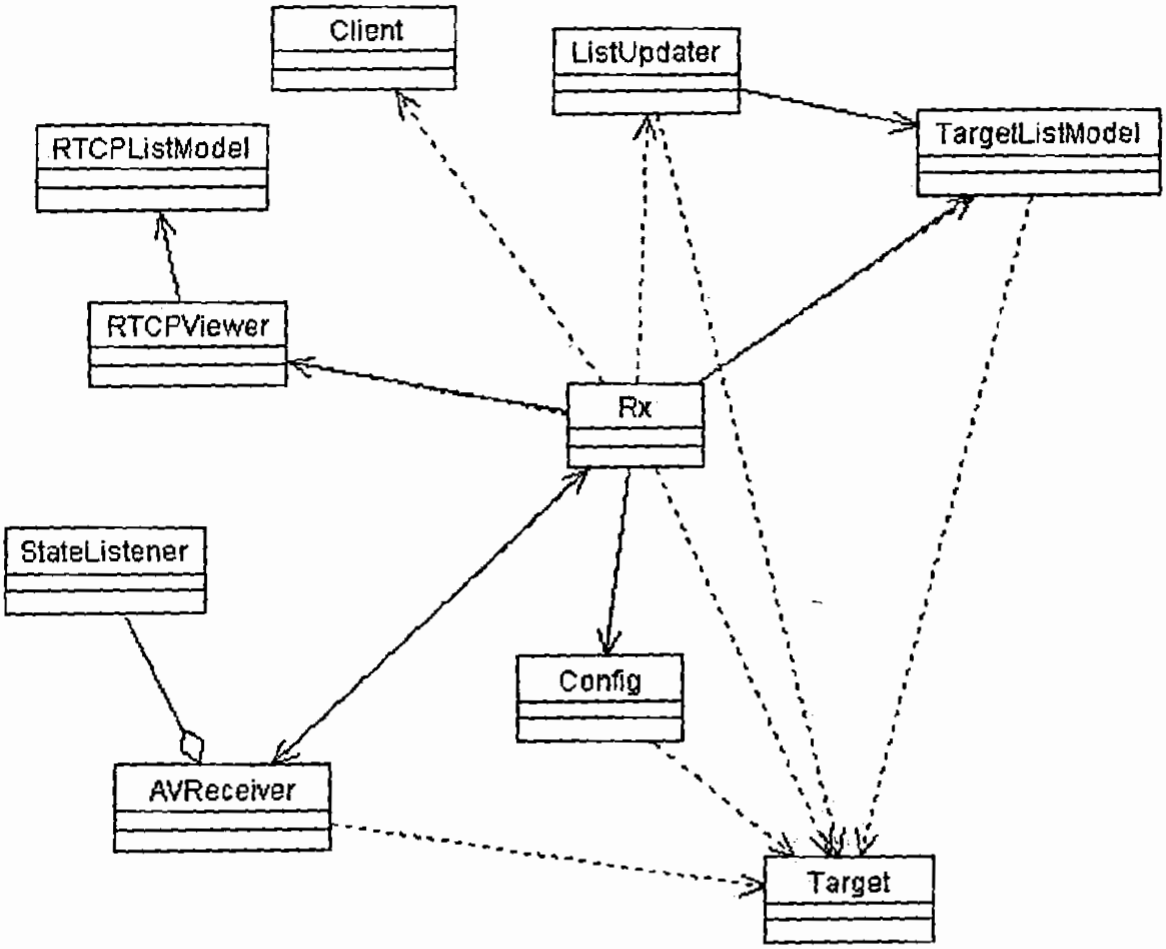


**Fig 3.10: Object Diagram of Receiver**

# CHAPTER 4

# DEVELOPMENT

# 4. Class Description

Class description is used to represent the overview of the class, its variables and methods as shown in the following tables:

**Table 4.1: Description of class TranSecureServ**

| TranSecureServ<br><br>This is the main class on transmitter side that creates GUI and calls other relevant classes. This class also takes input from the user i.e. the target IP and ports, media input device and cipher used to encrypt the data. | |
|---|---|
| **Variables** | **Remarks** |
| avTrasmitter | Object of class myTransmitter that is used to perform the transmission function. |
| ListModel | Stores the list of receivers. |
| cipher_selected | String variable that stores the name of selected cipher. |
| Config | Object of class Config |
| Sock | Datagram socket that transmits to receiver the name of selected cipher |
| Targets | Vector variable that stores the list of target receivers in memory. |
| **Methods** | **Remarks** |
| main() | Main function that initiates the program. |
| actionPerformed() | Actionlistener that executes instructions when any event has occurred. |
| addTargetToList() | Adds user input target to the list that stores target values in memory. |
| removeNonBaseTargets() | Validates if the user input is really a valid receiver, removes the input from the list if it is not. |
| mouseClicked() | Initiated if a mouse event is called. |

**Table 4.2: Description of class AVTransmit3**

| AVTransmit3:<br><br>This is the main class that interacts with JMF. This class uses java media framework library its classes to create the transmitter. This class also sets the media codecs to be used and the format of the media. | |
|---|---|
| **Variables** | **Remarks** |
| locator | Instance of class javax.media.medialocator that converts hardware device address and passes to media processor. |
| ipAddress | IP value input by user as target receiver. |
| portBase | Port value input by user as target receiver. |
| local_data_port | Local port that transmits data. |
| processor | Object of javax.media.processor class. A processor is used to explicitly process or store the captured media data. |
| dataOutput | Object of class javax.media.protocol.DataSource. A DataSource is constructed using media locator that gives the address of the other device that helps to initiate the capture process. |
| rtpMgrs | Object of class javax.media.rtp.RTPManager. A manager is used to coordinate an RTP session. It also keeps track of session participants and streams that are being transmitted. |
| Methods | Remarks |
| start() | Starts the transmission. Returns null if the transmission started successfully. |
| createProcessor() | Creates a processor for specified media locator. It creates data source for that specific locator and creates a processor to handle the input media locator. Wait for |

| | |
|---|---|
| | it to configure. It also gets the tracks from the processor and sets media format and video size. It also gets the output data source of the processor. |
| createTransmitter() | Creates RTP session to transmit the output of the processor to the user input IP address and port. This method uses RTPManager API to create sessions for each media track of the processor. |
| stop() | Stops the transmission. |
| setJPEGQuality() | Sets the encoding quality of the default codec. |
| waitForState() | Initiates instructions for specific processor states. |
| controllerUpdate() | Method implemented by inner class StateListener used to notify if a controller event occur. |

**Table 4.3: Description of class RTPSocketAdapter**

| *RTPSocketAdapter:* It is an implementation of RTPConnector for UDP based socket. | |
|---|---|
| **Variable** | **Comments** |
| dataSock | Datagram Socket that transmit RTP data in a session |
| ctrlSock | Datagram Socket that transmit RTCP data in a session |
| Addr | InetAddress that will receive RTP data |
| Port | Port that will receive data |
| cipher_selected | The name of the cipher selected to encrypt data |
| dataInStrm | Object of inner class SockInputStream that is an inner class to implement an PushSourceStream for UDP based Socket the .object store the input stream to receive RTP data. |

| | |
|---|---|
| dataOutStrm | Object of inner class SockOutpuStream that is an inner class to implement an OutputDataStream for UDP based Socket. The object store the output stream to send RTP data |
| ctrlInStrm | Object of inner class SockInputStream that is an inner class to implement an PushSourceStream for UDP based Socket the .object store the input stream to receive RTCP data. |
| ctrlOutStrm | Object of inner class SockOutpuStream that is an inner class to implement an OutputDataStream for UDP based Socket. The object store the output stream to send RTCP data |
| **Methods** | **Comments** |
| getDataInputStream() | Returns an input stream to receive the RTP data |
| getDataOutputStream() | Returns an output stream to send the RTP data |
| getControlInputStream() | Returns an input stream to receive the RTCP data |
| getControlOutputStream() | Returns an output stream to send the RTCP data |
| close() | Close all the RTP, RTCP streams |
| setReceiveBufferSize() | Set the receive buffer size of the RTP data channel |
| getReceiveBufferSize() | Get the receive buffer size set on the RTP data channel |
| setSendBufferSize() | Set the send buffer size of the RTP data channel |
| getSendBufferSize() | Get the send buffer size set on the RTP data channel |
| getRTCPBandwidthFraction() | Return the RTCP bandwidth fraction |
| getRTCPSenderBandwidthFraction() | Return the RTCP sender bandwidth fraction |

**Table 4.4: Description of class SocketOutputStream::RTPSocketAdapter**

| SocketOutputStream:<br><br>It is an inner class of RTPSocketAdapter and implements OutputDataStream based on UDP based socket. | |
|---|---|
| **Variable** | **Comments** |
| sock | Datagram Socket that transmit RTP data in a session |
| addr | InetAddress that will receive RTP data |
| port | Port that will receive data |
| cipher_selected | The name of the cipher selected to encrypt data |
| **Methods** | **Comments** |
| write () | Write UDP packets on the underlying network |

**Table 4.5: Description of class SocketInputStream::RTPSocketAdapter**

| SocketInputStream:<br><br>It is an inner class of RTPSocketAdapter and implements PullSourceStream based on UDP based socket. | |
|---|---|
| **Variable** | **Comments** |
| sock | Datagram Socket that transmit RTP data in a session |
| addr | InetAddress that will receive RTP data |
| port | Port that will receive data |
| cipher_selected | The name of the cipher selected to encrypt data |
| sth | It is Object of SourceTransferHandler that needs to be implemented if implementing |

| | PullSourceStream |
|---|---|
| **Methods** | **Comments** |
| read () | Read UDP packets from the underlying network and passes to the RTP Manger |
| run() | Loop and notify the transfer handler of new data |

**Table 4.6: Description of class *Config***

| Config: Class that write the values stored in the vector containing list of receivers in a file "Xmit.dat" when the application window is closed. This class also retrieves the stored values from the file when the application initiates. | |
|---|---|
| **Variable** | **Comments** |
| local_data_port | String variable that store the local port number that transmits the data |
| media_locator | Media locator passed to the class |
| target | Vector containing list of receivers |
| **Methods** | **Comments** |
| addTarget() | Add any new receiver added to its vector |
| read() | Read the receivers from file |
| write () | Write the receiver list in file |

**Table 4.7: Description of class Target**

| Target: Class that sets the ip and port and referenced from different classes | |
|---|---|
| **Variable** | **Comments** |
| ip | ip passed to the class |
| port | Port passed to the class |

| localport | contain local port number that transmit data |
|---|---|
| **Methods** | **Comments** |
| Target() | Constructor of the class set the variable |

**Table 4.8: Description of class CipherSelect**

| CipherSelect:<br><br>Class that initiate the cipher and sets key for the corresponding selected cipher name | |
|---|---|
| **Variable** | **Comments** |
| c | Cipher class object |
| cipher_selected | name of cipher selected by user |
| K | Object of java.security.Key |
| Kg | Object of javax.crypto.KeyGenerator |
| **Methods** | **Comments** |
| CipherSelect() | Constructor of the class set the name of selected cipher |
| Init () | Initialize the cipher according to the selected cipher |

**Table 4.9: Description of class AVReceive3**

| AVReceive3:<br><br>This is the main class that interacts with JMF. This class uses java media framework library and uses its classes to create the receiver. This class also sets the media codecs to be used and the format of the media. The class also receives RTP transmission using the RTP connector. | |
|---|---|
| **Variables** | **Comments** |
| mgrs | Object of class javax.media.rtp.RTPManager. A manager is used to coordinate an RTP session. It also |

| | |
|---|---|
| | keeps track of session participants and streams that are being transmitted. |
| playerWindows | GUI class for the player. A player is used to display the received media. It processes an input stream of media data and renders it at a precise time. A DataSource is used to deliver the input media-stream to the Player. |
| sessions | Stream variable that stores the session IP and port address given as input by the user. |
| **Methods** | **Comments** |
| initialize() | This method creates instance of RTPmanagers and playerwindows. Opens RTP session. Initializes RTPManager with RTPSocketAdapter |
| controllerUpdate() | Implementation of controllerListener that listens for player related events. |
| update(ReceiveStreamEvent) | Implements ReceiveStreamListener that listens for the media stream related events. |
| update(SessionEvent) | Implements SessionEvent that listens for the session related events. |
| close() | Closes the players and the session managers. |
| find() | Checks and initiates whether a player is created for received stream. |

## 4.1 Implementation

In this chapter we shall discuss the implementation of the system. We shall discuss the technologies used to develop the system and the benefits these technologies. These technologies have greater scope on other related technologies. In section 4.2 we shall discuss JAVA and in the next section i.e. 4.3 JMF.

## 4.2 JAVA

Java as a language needs no introduction. It's never been out of the news since it was released. The Java2 platform is a fast maturing way to program portable, object oriented, secure, and internet ready application. Over the last two years, Java's support for application development has expanded enormously. The Java APIs in question have very broad industry support, having been developed by java soft in wide consultation with expert partners. In consequence the Java revolution of portable code and open APIs is married with an evolution in existing products. The wide ability of products to run java application on the server has made this a fast moving and very competitive market, but essential compatibility through specification, standard APIs and class libraries has held. This makes server side Java a very exciting area.

Although Java has an enormous number of products that support solving different types of problems. Some of those technologies are listed below.

1. Java Media Framework (JMF)
2. Java Naming and Directory Interface (JNDI)
3. Java Secure Socket Extension (JSSE)
4. Java Speech API
5. Java 3D API
6. CORBA
7. Enterprise JavaBeans

8. JavaMail
9. Java Message Service (JMS)
10. Java Servlets
11. Java Advanced Imaging
12. Java Media APIs
13. Java Embedded Server Technology

Discussion on all these topics will become very lengthy therefore we will discuss only Java Media Framework in the coming section as this is used for developing real-time application for video conferencing.

## 4.3 Java Media Framework

The Java™ Media Framework (JMF) is an application programming interface (API) for incorporating time-based media into Java applications and applets. It is intended for Java programmers who want to incorporate time-based media into their applications and for technology providers who are interested in extending JMF and providing JMF plug-ins to support additional media types and perform custom processing and rendering.

The JMF 1.0 API (the Java Media Player API) enabled programmers to develop Java programs that presented time-based media. The JMF 2.0 API extends the framework to provide support for capturing and storing media data, controlling the type of processing that is performed during playback, and performing custom processing on media data streams. In addition, JMF 2.0 defines a plug-in API that enables advanced developers and technology providers to more easily customize and extend JMF functionality [10].

The following classes and interfaces are new in JMF:

| | | |
|---|---|---|
| AudioFormat | BitRateControl | Buffer |
| BufferControl | BufferToImage | BufferTransferHandler |

| CaptureDevice | CaptureDeviceInfo | CaptureDeviceManager |
|---|---|---|
| CloneableDataSource | Codec | ConfigureCompleteEvent |
| ConnnectionErrorEvent | DataSink | DataSinkErrorEvent |
| DataSinkEvent | DataSinkListener | Demultiplexer |
| Effect | EndOfStreamEvent | FileTypeDescriptor |
| Format | FormatChangeEvent | FormatControl |
| FrameGrabbingControl | FramePositioningControl | FrameProcessingControl |
| FrameRateControl | H261Control | H261Format |
| H263Control | H263Format | ImageToBuffer |
| IndexedColorFormat | InputSourceStream | KeyFrameControl |
| MonitorControl | MpegAudioControl | Multiplexer |
| NoStorageSpaceErrorEvent | PacketSizeControl | PlugIn |
| PlugInManager | PortControl | Processor |
| ProcessorModel | PullBufferDataSource | PullBufferStream |
| PushBufferDataSource | PushBufferStream | QualityControl |
| Renderer | RGBFormat | SilenceSuppressionContr ol |
| StreamWriterControl | Track | TrackControl |
| VideoFormat | VideoRenderer | YUVFormat |

In addition, the MediaPlayer Java Bean has been included with the JMF API in javax.media.bean.playerbean. MediaPlayer can be instantiated directly and used to present one or more media streams.

Future versions of the JMF API will provide additional functionality and enhancements while maintaining compatibility with the current API.

### 4.3.1 Session Manager

In JMF, a SessionManager is used to coordinate an RTP session. The session manager keeps track of the session participants and the streams that are being transmitted.

The session manager maintains the state of the session as viewed from the local participant. In effect, a session manager is a local representation of a distributed entity, the RTP session. The session manager also handles the RTCP control channel, and supports RTCP for both senders and receivers.

The SessionManager interface defines methods that enable an application to initialize and start participating in a session, remove individual streams created by the application, and close the entire session

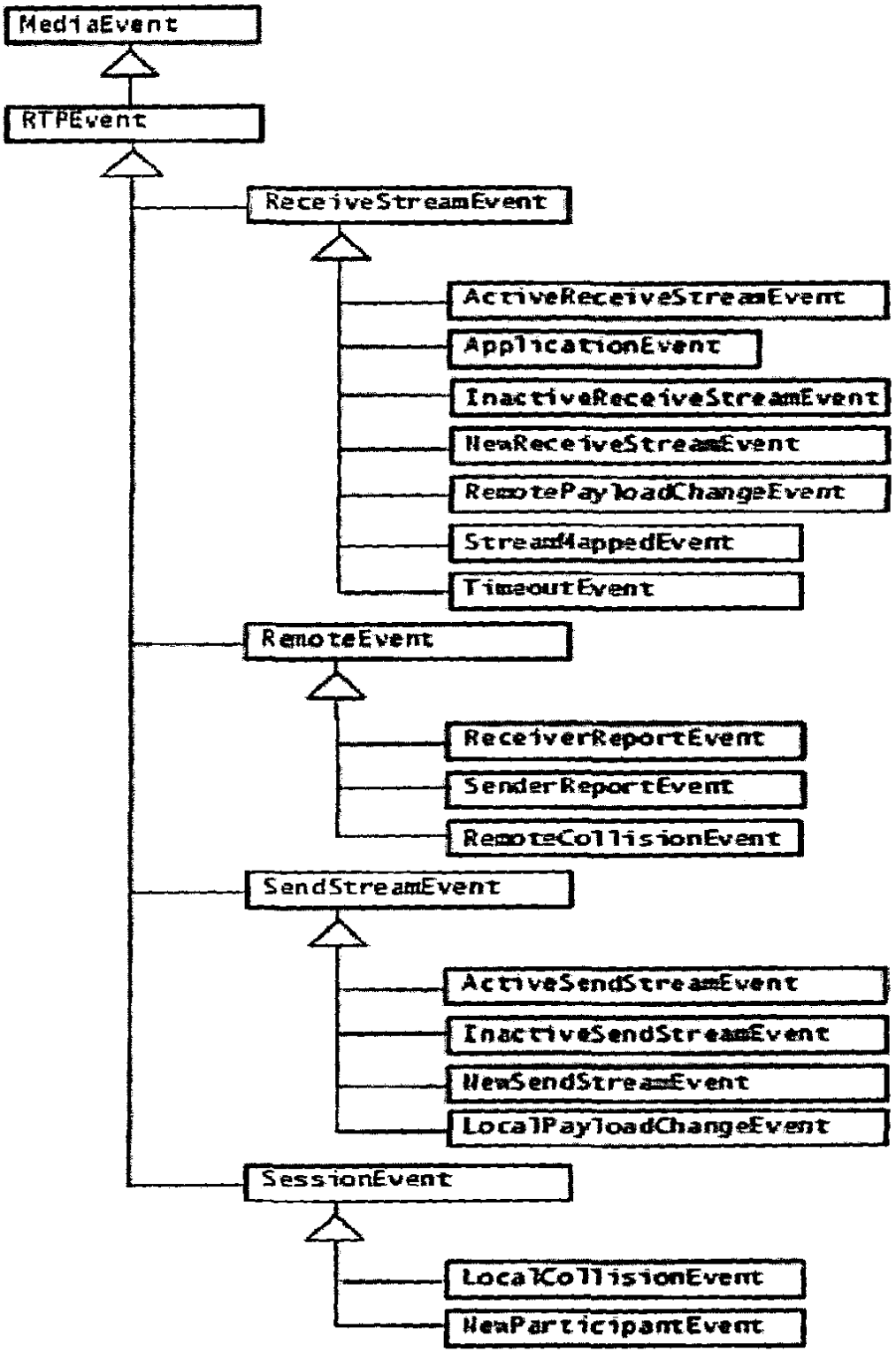


**Figure 4.1: Events in JMF**

## 4.4 Code Sample for Transmitting and Receiving Audio and Video of Client using RTP

Here is the sample code for transmitting and receiving audio and video. They are Transmitter for video transmission, Receiver for video receiver, AudioReceiver for audio reception and AudioTransmission for audio transmission.

### 4.4.1 Classes and Packages for Media Streaming and Media Reception

Following are the packages and classes used for media streaming and reception:

1. import java.awt.*
2. import java.io.*
3. import java.net.InetAddress
4. import javax.media.*
5. import javax.media.protocol.*
6. import javax.media.format.*
7. import javax.media.control.*
8. import javax.media.rtp.*
9. import javax.media.rtp.rtcp.*
10. import com.sun.media.rtp.*
11. import java.util.*
12. import javax.swing.*

### 4.4.2 Classes and Packages used for Encryption

Following are the packages and classes used for media encryption:

1. import javax.crypto.*;
2. import javax.security.*;

You can use JMF to capture media data from a capture device such as a microphone or video camera. Captured media data can be processed and rendered or stored for future use.

To capture media data, the following tasks are performed:

1. Locate the capture device you want to use by querying the CaptureDeviceManager.

   ```
   CaptureDeviceManager cd = new CaptureDeviceManager();

   YUVFormat yuvformat = new YUVFormat();
   ```

2. Get a CaptureDeviceInfo object for the device.

   ```
   dl = cd.getDeviceList(yuvformat);

   CaptureDeviceInfo di = (CaptureDeviceInfo)dl.firstElement();
   ```

3. You access capture devices through the CaptureDeviceManager. The CaptureDeviceManager is the central registry for all of the capture devices available to JMF. You can get a list of the available capture devices by calling the CaptureDeviceManager.getDeviceList method.

   Each device is represented by a CaptureDeviceInfo object. To get the CaptureDeviceInfo object for a particular device, you call CaptureDeviceManager.getDevice:

   ```
   CaptureDeviceInfo

   deviceInfo = CaptureDeviceManager.getDevice("deviceName");
   ```

4. Get a MediaLocator from the CaptureDeviceInfo object and use it to create a DataSource.

```
this.locator = di.getLocator();
```

5. To capture media data from a particular device, you need to get the device's MediaLocator from its CaptureDeviceInfo object. You can either use this MediaLocator to construct a Player or Processor directly, or use the MediaLocator to construct a DataSource that you can use as the input to a Player or Processor. To initiate the capture process, you start the Player or Processor.

```
DataSource ds;

DataSource clone;

try {

  ds = javax.media.Manager.createDataSource(locator);

} catch (Exception e) {

  return "Couldn't create DataSource";

}
```

6. When you use a capture DataSource with a Player, you can only render the captured media data. To explicitly process or store the captured media data, you need to use a Processor. Create a Player or Processor using the DataSource.

```
// Create a processor for the specified media locator

// and program it to output JPEG/RTP

  // Try to create a processor to handle the input media locator
```

```
try {
    processor = javax.media.Manager.createProcessor(ds);
    processor.addControllerListener( this);
} catch (NoProcessorException npe) {
    return "Couldn't create processor";
} catch (IOException ioe) {
    return "IOException creating processor";
}
```

7. Start the Processor to begin the capture process

```
// Create an RTP session to transmit the output of the
// processor to the specified IP address and port no.
    result = createTransmitter();
```

### 4.4.3 Configuring a Processor

In addition to the Realizing and Prefetching phases that any Player moves through as it prepares to start, a Processor also goes through a Configuring phase. You call configure to move an Unrealized Processor into the Configuring state.

While in the Configuring state, a Processor gathers the information it needs to construct TrackControl objects for each track. When it's finished, it moves into the Configured

state and posts a ConfigureCompleteEvent. Once a Processor is Configured, you can set its output format and TrackControl options. When you're finished specifying the processing options, you call realize to move the Processor into the Realizing state and begin the realization process.

Once a Processor is realized, further attempts to modify its processing options are not guaranteed to work. In most cases, a FormatChangeException will be thrown.

```
DataSource ds;

DataSource clone;

try {

  ds = javax.media.Manager.createDataSource(locator);

} catch (Exception e) {

  return "Couldn't create DataSource";

}

// Try to create a processor to handle the input media locator

try {

  processor = javax.media.Manager.createProcessor(ds);

  processor.addControllerListener( this);

} catch (NoProcessorException npe) {

  return "Couldn't create processor";

} catch (IOException ioe) {
```

```
    return "IOException creating processor";
}

// Wait for it to configure
boolean result = waitForState(processor, Processor.Configured);
if (result == false)
    return "Couldn't configure processor";
```

### 4.4.4 Retrieving the Processor Output

Once the format of a Processor's track has been set and the Processor has been realized, the output DataSource of the Processor can be retrieved. You retrieve the output of the Processor as a DataSource by calling getDataOutput. The returned DataSource can be either a PushBufferDataSource or a PullBufferDataSource, depending on the source of the data.

The output DataSource is connected to the SessionManager using the createSendStream method. The session manager must be initialized before you can create the send stream.

If the DataSource contains multiple SourceStreams, each SourceStream is sent out as a separate RTP stream, either in the same session or a different session. If the DataSource contains both audio and video streams, separate RTP sessions must be created for audio and video. You can also clone the DataSource and send the clones out as different RTP streams in either the same session or different sessions.

```
// Get the tracks from the processor
TrackControl [] tracks = processor.getTrackControls();

// Do we have atleast one track?
if (tracks == null || tracks.length < 1)
   return "Couldn't find tracks in processor";

// Set the output content descriptor to RAW_RTP
// This will limit the supported formats reported from
// Track.getSupportedFormats to only valid RTP formats.
ContentDescriptor cd = new ContentDescriptor(ContentDescriptor.RAW_RTP);
processor.setContentDescriptor(cd);
Format supported[];
Format chosen;
boolean atLeastOneTrack = false;
// Program the tracks.
for (int i = 0; i < tracks.length; i++) {
   Format format = tracks[i].getFormat();
   if (tracks[i].isEnabled()) {
```

```
supported = tracks[i].getSupportedFormats();
// We've set the output content to the RAW_RTP.
// So all the supported formats should work with RTP.
// We'll just pick the first one.

if (supported.length > 0) {
    if (supported[0] instanceof VideoFormat) {
        // For video formats, we should double check the
        // sizes since not all formats work in all sizes.
        chosen = checkForVideoSizes(tracks[i].getFormat(),
                                    supported[0]);
    } else
        chosen = supported[0];
    tracks[i].setFormat(chosen);
    System.err.println("Track " + i + " is set to transmit as:");
    System.err.println("  " + chosen);
    atLeastOneTrack = true;
} else
```

```
            tracks[i].setEnabled(false);

        } else

            tracks[i].setEnabled(false);

    }
```

### 4.4.5 Creating a RTP Player for Each New Received Stream

To play all of the ReceiveStreams in a session, you need to create a separate Player for each stream. When a new stream is created, the session manager posts a NewReceiveStreamEvent. Generally, you register as a ReceiveStreamListener and construct a Player for each new ReceiveStream. To construct the Player, you retrieve the DataSource from the ReceiveStream and pass it to Manager.createPlayer.

To create a Player for each new receive stream in a session:

1. Set up the RTP session:

    - Create a SessionManager. For example, construct an instance of com.sun.media.rtp.RTPSessionMgr. (RTPSessionMgr is an implementation of SessionManager provided with the JMF reference implementation.)
    - Call RTPSessionMgr addReceiveStreamListener to register as a listener.
    - Initialize the RTP session by calling RTPSessionMgr initSession.
    - Start the RTP session by calling RTPSessionMgr startSessio

```
PushBufferDataSource pbds = (PushBufferDataSource)dataOutput;

PushBufferStream pbss[] = pbds.getStreams();
```

```
rtpMgrs = new RTPManager[pbss.length];

localPorts = new int[ pbss.length];

SessionAddress localAddr, destAddr;

InetAddress ipAddr;

SendStream sendStream;

int port;

SourceDescription srcDesList[];


for (int i = 0; i < pbss.length; i++) {

// for (int i = 0; i < 1; i++) {

  try {

      rtpMgrs[i] = RTPManager.newInstance();


      port = local_data_port + 2*i;


      localPorts[ i]= port;


      localAddr = new SessionAddress( InetAddress.getLocalHost(),

                                      port);
```

```
                rtpMgrs[i].initialize( localAddr);

                rtpMgrs[i].addReceiveStreamListener(this);

                rtpMgrs[i].addRemoteListener(this);

            for( int k= 0; k < targets.size(); k++) {

                Target target= (Target) targets.elementAt( k);

                int targetPort= new Integer( target.port).intValue();

                addTarget( localPorts[ i], rtpMgrs[ i], target.ip, targetPort + 2*i);

    }

                sendStream = rtpMgrs[i].createSendStream(dataOutput, i);

                sendStream.start();

        } catch (Exception e) {

                e.printStackTrace();



                return e.getMessage();

        }

    }
```

### 4.4.6 Implementing Controller Listener

To implement the ControllerListener interface, you need to:

1. Implement the ControllerListener interface in a class.
2. Register that class as a listener by calling addControllerListener on the Controller that you want to receive events from.

When a Controller posts an event, it calls controllerUpdate on each registered listener.

Typically, controllerUpdate is implemented as a series of if-else statements.

This filters out the events that you are not interested in. If you have registered as a listener with multiple Controllers, you also need to determine which Controller posted the event. ControllerEvents come "stamped" with a reference to their source that you can access by calling getSource.

When you receive events from a Controller, you might need to do some additional processing to ensure that the Controller is in the proper state before calling a control method. For example, before calling any of the methods that are restricted to Stopped Players, you should check the Player object's target state by calling getTargetState. If start has been called, the Player is considered to be in the Started state, though it might be posting transition events as it prepares the Player to present media.

Some types of ControllerEvents contain additional state information. For example, the StartEvent and StopEvent classes each define a method that allows you to retrieve the media time at which the event occurred

```
public void controllerUpdate( ControllerEvent ce) {

    System.out.println( ce);

    if( ce instanceof DurationUpdateEvent) {
```

```
            Time duration= ((DurationUpdateEvent) ce).getDuration();


            System.out.println( "duration: " + duration.getSeconds());
        } else if( ce instanceof EndOfMediaEvent) {
            System.out.println( "END OF MEDIA - looping=" + looping);
            if( looping) {
                processor.setMediaTime( new Time( 0));
                    processor.start();
            }
        }
    }
```

### 4.4.7 Displaying Media Interface Components

A Player generally has two types of user interface components, a visual component and a control-panel component. Some Player implementations can display additional components, such as volume controls and download-progress bars.

### 4.4.8 Displaying a Visual Component

A visual component is where a Player presents the visual representation of its media, if it has one. Even an audio Player might have a visual component, such as a waveform display or animated character.

To display a Player object's visual component, you:

1. Get the component by calling getVisualComponent.
2. Add it to the applet's presentation space or application window.

You can access the Player object's display properties, such as its $x$ and $y$ coordinates, through its visual component. The layout of the Player components is controlled through the AWT layout manager.

### 4.4.9 Displaying a Control Panel Component

A Player often has a control panel that allows the user to control the media presentation. For example, a Player might be associated with a set of buttons to start, stop, and pause the media stream, and with a slider control to adjust the volume.

Every Player provides a default control panel. To display the default control panel:

1. Call getControlPanelComponent to get the Component.
2. Add the returned Component to your applet's presentation space or application window.

If you prefer to define a custom user-interface, you can implement custom GUI Components and call the appropriate Player methods in response to user actions. If you register the custom components as ControllerListeners, you can also update them when the state of the Player changes.

# CHAPTER 5

# TESTING

# 5. Testing

System testing is an essential step for the development of a reliable and error-free system. Testing is the process of executing a program with the explicit intention of finding errors i.e., making the program fail and test cases are devised with the purpose in mind. A test case is a set of data items that the system processes as normal input. A successful test is the one that finds an error.

## 5.1 Testing Strategies

The basic strategies that were used for testing were following:

1. Specification testing
2. Black box testing
3. White box testing
4. Regression testing
5. Acceptance testing
6. Assertion testing
7. Unit testing
8. System testing

Each of the testing schemes is discussed below.

### 5.1.1 Specification Testing

Even if the code testing is performed exclusively, it doesn't provide grantee against the program failure. Code testing doesn't answer whether the code meets the agreed specification document. It doesn't also determine whether all aspects of the design are implemented.

Therefore, examining specifications stating what program should do and how it should behave under various conditions performs specification testing. Test cases are developed

to test the range of values expected including both valid and invalid data. It helps in finding discrepancies between the system and its original objective. During this testing phases, all efforts were made to remove programming bugs and minor design faults.

### 5.1.2 Black Box Testing

In Black Box only the functionality was tested without any regard to the code written. If the functionality, which was expected from a component, is provided then black box testing is completed.

### 5.1.3 White Box Testing

In White Box testing internal code written in every component was tested and it was checked that the code written is efficient in utilizing resources of the system like memory, bandwidth, or the utilization of input output.

### 5.1.4 Regression Testing

In regression testing the software was testing against the boundary condition. Various input fields were tested against abnormal values and it was tested that the software does not behave abnormally at any time.

### 5.1.5 Acceptance Testing

In acceptance testing the software was tested for its completeness that it is ready. Normally the quality assurance department performs the acceptance testing that the software is ready and can be exported.

### 5.1.6 Assertion Testing

In assertion testing the software is tested against the possible assertions. Assertions are used to check the program and various locations that whether the state of the program at a particular point is the same as expected or not.

### 5.1.7 Unit Testing

In unit testing we checked that all the individual components were working properly. Before integration of the entire components unit testing is essential because it gives a confidence that all the components individually are working fine and ready to be integrated with other ones.

### 5.1.8 System Testing

When all the units were working properly and unit testing was performed then comes the time for system testing where we checked all the integrated components as a whole and looked for possible discrepancies, which could have arisen after the integration.

## 5.2 System Evaluation

The objectives of the system evaluation are to determine whether the desired objectives have been accomplished or not. Determining the merits and demerits of the proposed system over the existing system is also covered in the system evaluation. This is concerned with the detailed study of the developed system, from implementation point of view. At the end, some suggestions for the improvements of the system are coded.

## 5.3 Testing TranSecure

Testing process of TranSecure started as different modules were completed. Our project is mainly divided in to three parts on the basis of technology. The interfaces were tested as they were built. As two interfaces were directly related to human input so unit testing was applied to them besides Regression testing. These interfaces were tested with different data sets and the syntax errors were removed. Then some validation checks were introduced in to the code for other interfaces as well. By applying the extensive tests to the video and audio transmission and receiving mechanism we tried to achieve the best ever quality.

Upon successful testing of individual modules the application was tested as a whole. Syntax changes were made where ever required. Then finally all the links in the application were updated and views of all the users were changed to new requirements. In the end all the checks for debugging purpose were removed and the application's interface was once again updated and minor changes were made as well.

## 5.4 Test Results

The efficiency of different algorithms was noticed for different test files of different sizes

File Sizes:

- 10 KB
- 100 KB
- 1 MB

**Table 5.1: Test Table for 10 KB Test File**

| | Null | DES | 3DES | AES |
|---|---|---|---|---|
| Run 1 | 91 | 120 | 90 | 92 |
| Run 2 | 91 | 101 | 100 | 97 |
| Run 3 | 90 | 100 | 100 | 93 |
| Average | 90.66 | 107 | 96.66 | 94 |

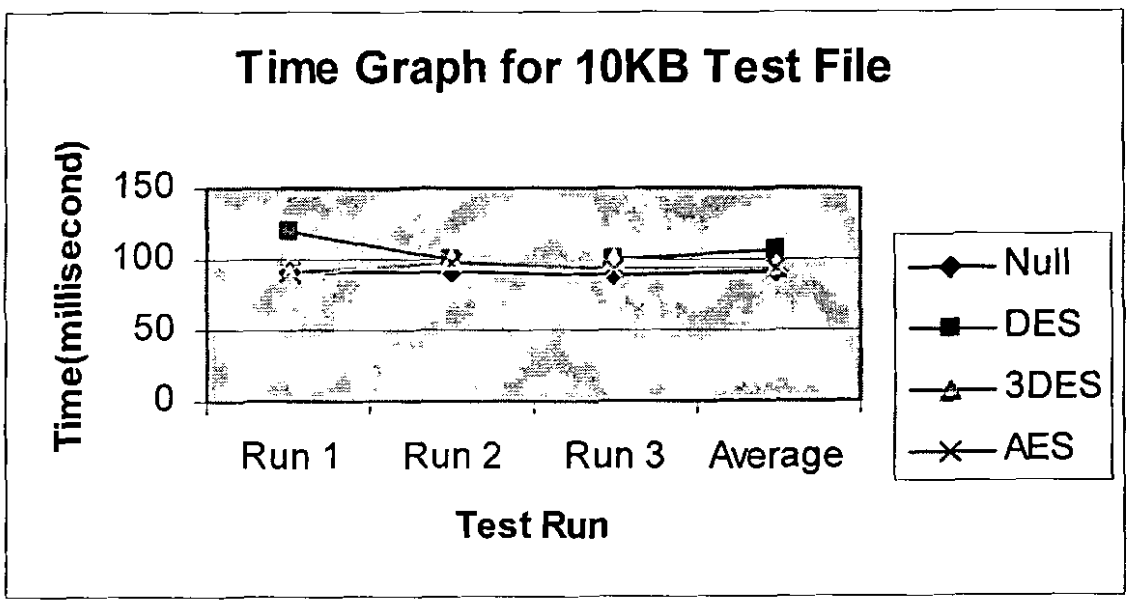


**Figure 5.1: Time Graph for 10 KB test file**

**Table 5.2: Test Table for 100 KB Test File**

| | Null | DES | 3DES | AES |
|---|---|---|---|---|
| Run 1 | 403 | 521 | 461 | 411 |
| Run 2 | 411 | 460 | 480 | 431 |
| Run 3 | 407 | 471 | 490 | 430 |
| Average | 407 | 484 | 477 | 424 |

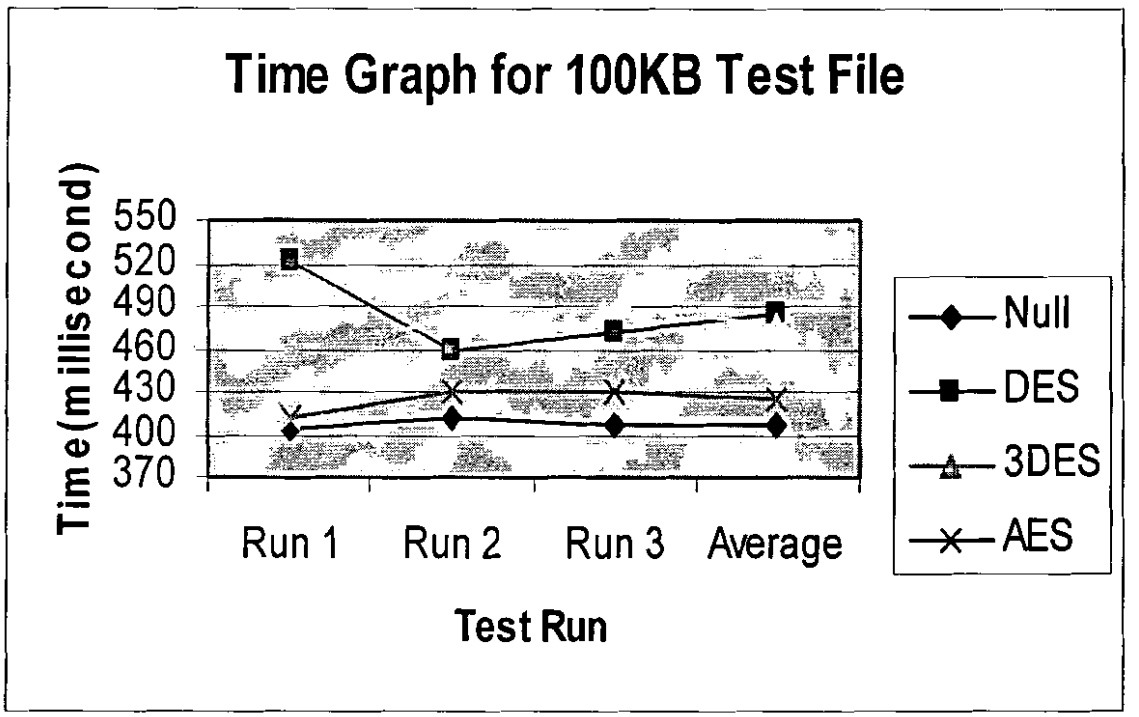


**Figure 5.2: Time Graph for 100 KB test file**

**Table 5.3: Test Table for 1 MB Test File**

| | Null | DES | 3DES | AES |
|---|---|---|---|---|
| Run 1 | 390 | 1352 | 802 | 791 |
| Run 2 | 490 | 1022 | 801 | 771 |
| Run 3 | 440 | 1081 | 752 | 751 |
| Average | 440 | 1151.667 | 785 | 771 |

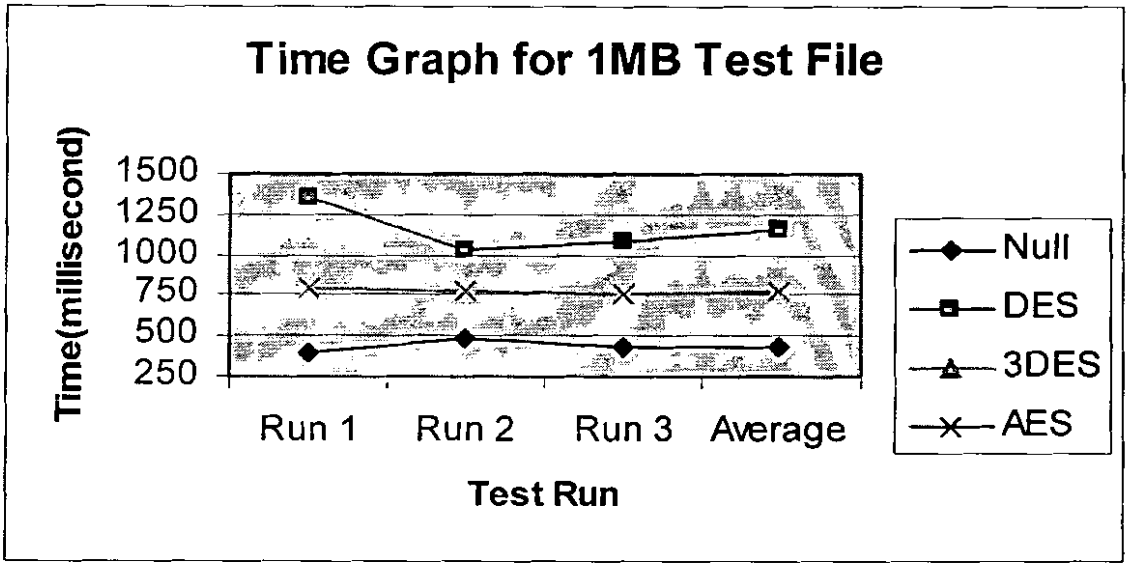


**Figure 5.3: Time Graph for 1 MB test file**

**Table 5.4: Comparison Table of different Algorithms**

| | Null | DES | 3DES | AES |
|---|---|---|---|---|
| 10KB | 90.66 | 107 | 96.66 | 94 |
| 100KB | 407 | 484 | 477 | 424 |
| 1MB | 440 | 1151.67 | 785 | 771 |

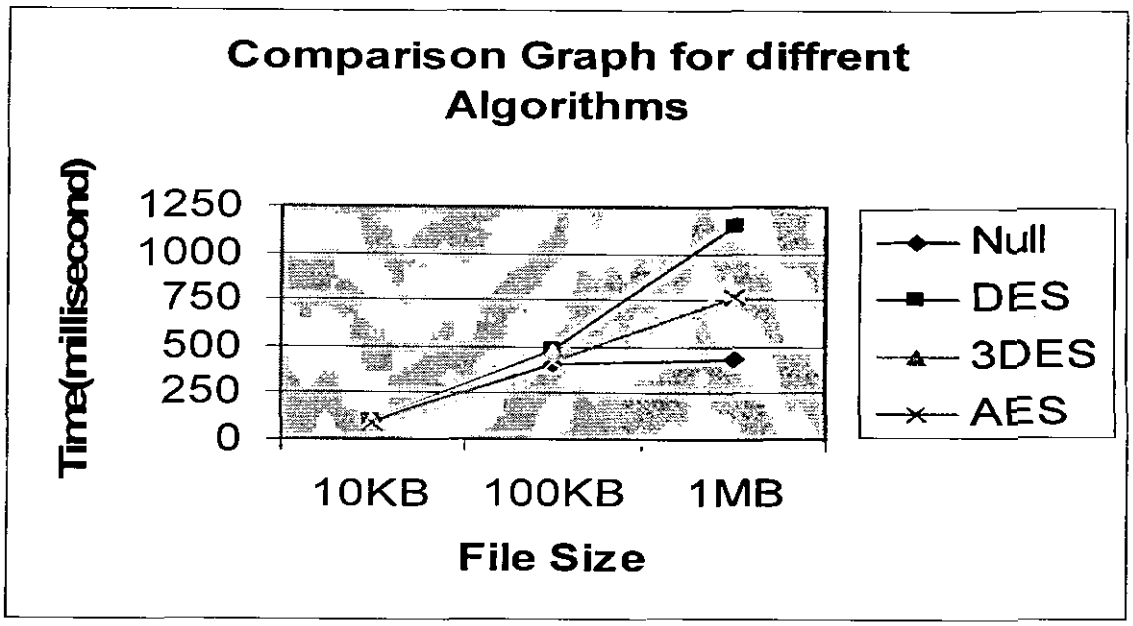


**Figure 5.4: Comparison Graph for Different Algorithms**

# CHAPTER 6

# USER MANUAL

# 6. Introduction

TranSecure has been designed keeping in view user's interaction and ease in use. All interfaces are simple and easy to use. This user manual facilitates the user to understand different forms and interfaces. The use of forms and different options are described in details in this manual.

## 6.1 User Manual for Transmitter

User manual provide relevant information for usage of this software.

### 6.1.1 Add Receiving Client

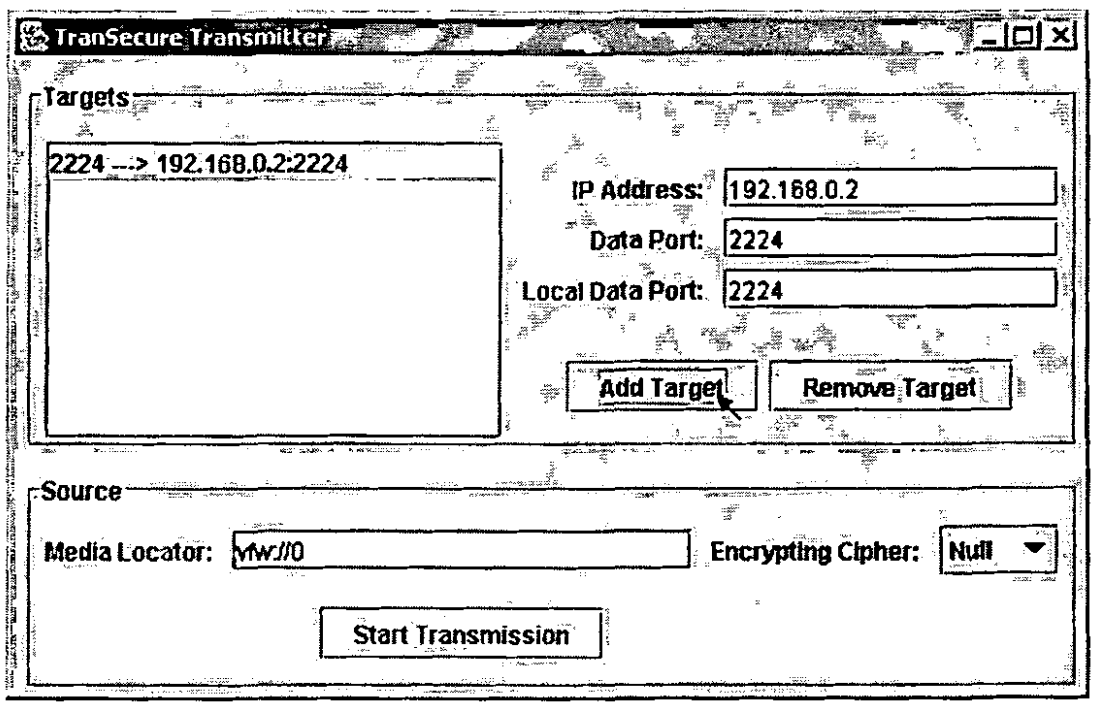


**Figure 6.1: Add Receiving client**

The transmitting module adds target clients to its list. To add a target receiver, the IP address and port of the receiving terminal along with the local port that is transmitting data should be inserted and then the "Add Target" button be pressed. A target receiver will be added to the list of receivers as our application supports the multicast transmission.

The string 2224 ---> 192.168.0.2:2224 tells that the local port 2224 is transmitting payload to target client 192.168.0.2 port 2224.

A log of the target client list is stored in a file "Xmit.dat" that reloads the list in the program when restarted.

## 6.1.2 Remove Receiving Client

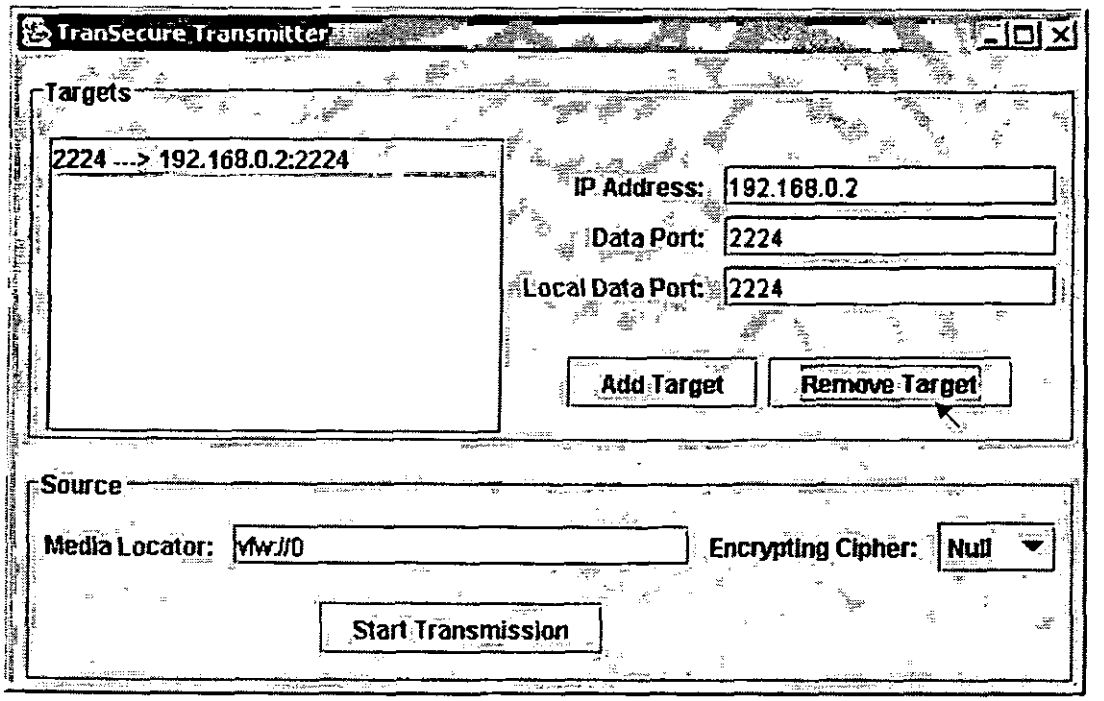


**Fig 6.2: Remove Receiving Client**

The transmitting module removes target clients from its list. To remove a target receiver, an address must be selected and the "Remove Target" is pressed. When the button is clicked it removes the selected address. If any address is not selected, the button "Remove Target" returns nothing and no address is removed.

The removed address will also be removed not only from the target list but also for the log file "Xmit.dat".

The status of the application after removing the target address is shown in the following figure.

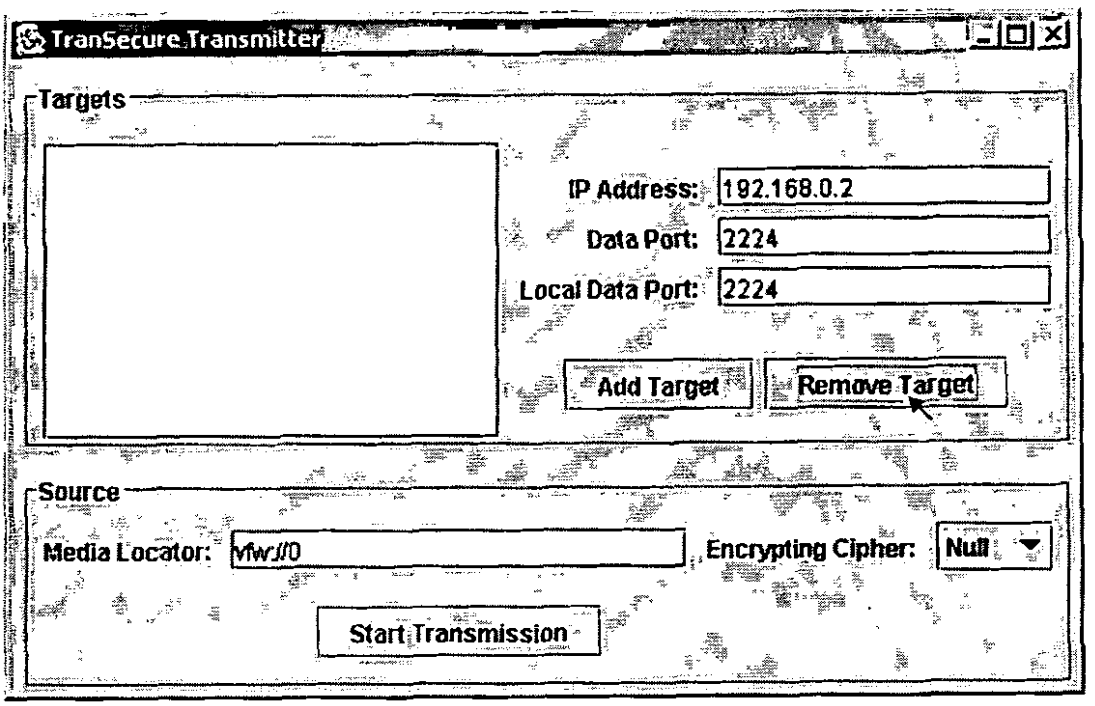


**Figure 6.3: After pressing the "Remove Target", this form will appear**

### 6.1.3 The Media Locator

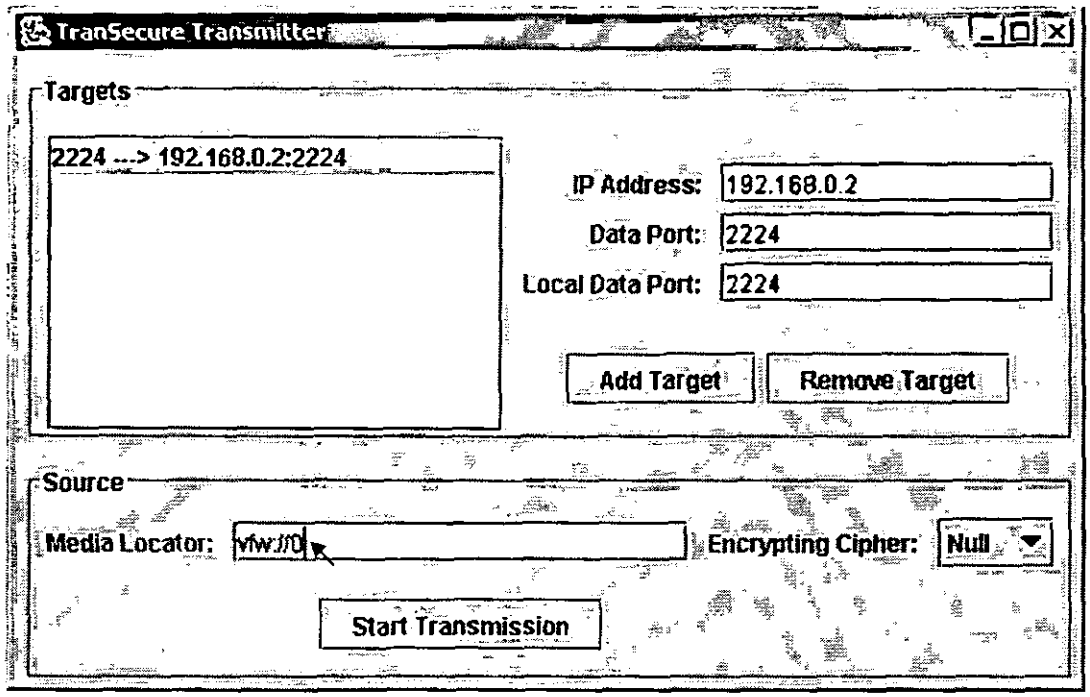


**Fig 6.4: Media Locator**

The Media Locator describes the location of data source. For example, in our case the vfw://0 the data source is a camera attached to the USB port and live streaming is being done. For any recorded media, file:/C:\media1.dat would transmit the file media1.dat located on path C:\.

The media stream will test the payload and make as many sessions for each stream. For example, if any file has both audio and video, the each stream will have separate session on separate ports with the receiver.

### 6.1.4 Encrypting Payload

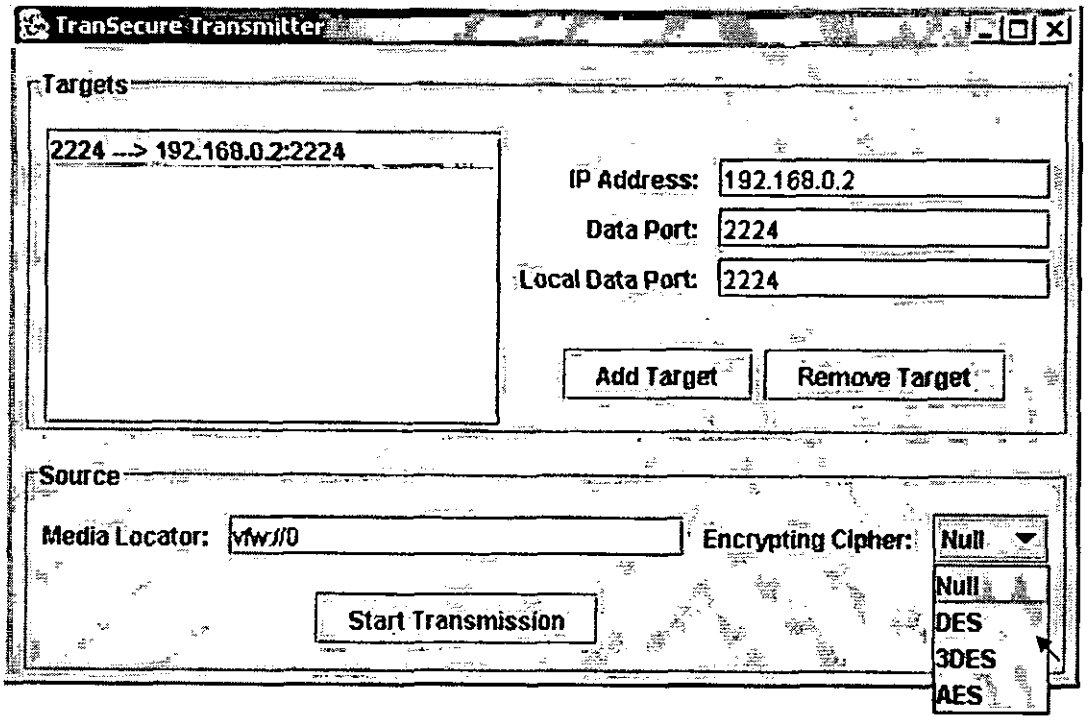


**Fig 6.5: Encrypting Payload**

The application provides support for the three encryption algorithms that are proposed by the RFC 3350 which is RFC for Real-time Transport protocol. The three algorithms DES, 3DES and AES are the standard algorithms that can encrypt the payload on the underlying protocol.

### 6.1.5 Start Transmission

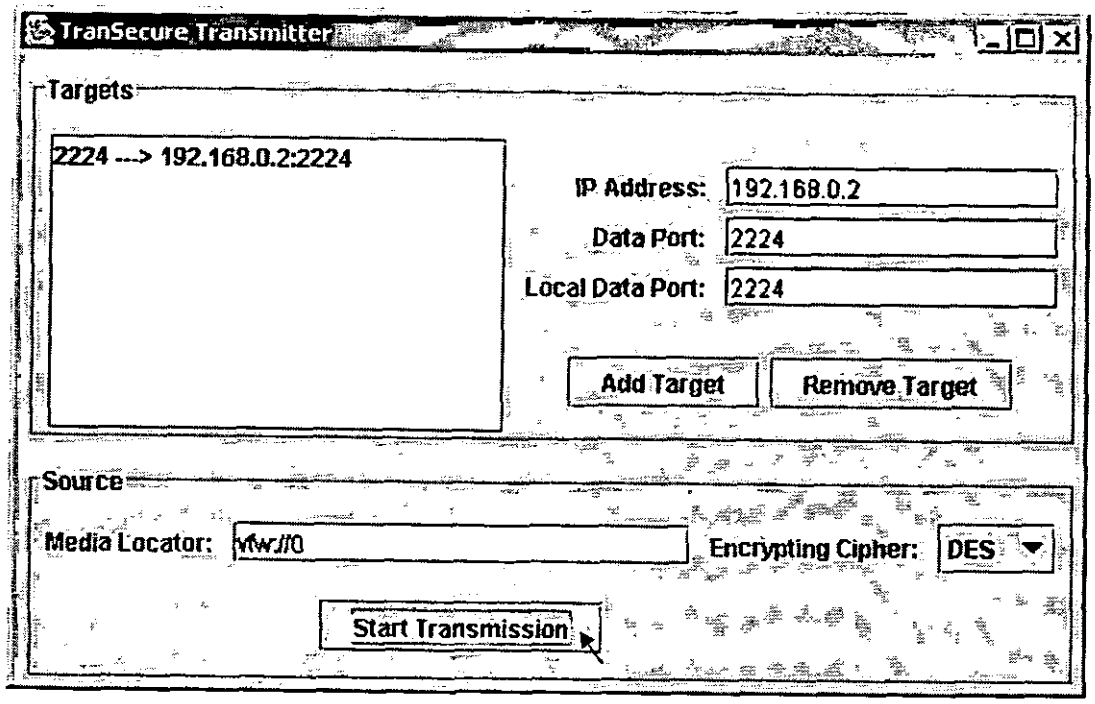


**Figure 6.6: Start Transmission**

After the destination addresses are added to the list, the encryption cipher is selected, the system is ready to transmit the payload across the network to the listed receivers.

The system will capture data from the device or from the media file whose address is provided. The media stream will test the payload and make as many sessions for each stream. For example, if any file has both audio and video, the each stream will have separate session on separate ports with the receiver.

### 6.1.6 Stop Transmitter

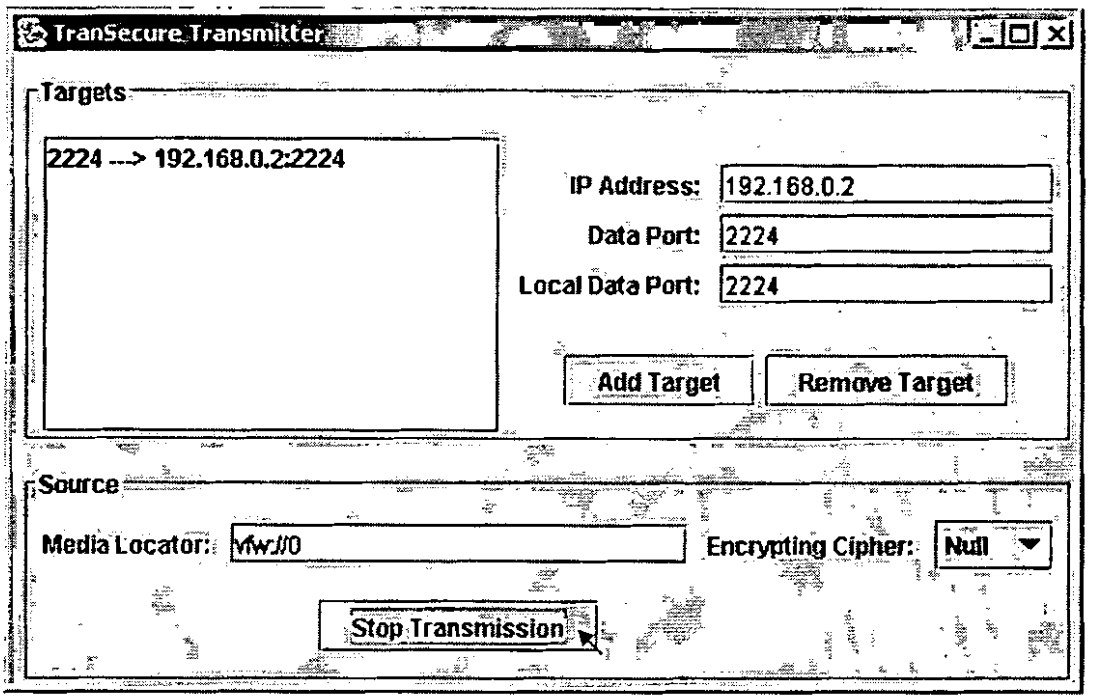


**Figure 6.7: Stop Transmission**

After the successful transmission of the data is started, the application let the user to stop the transmission. The transmission to all the receivers will be stopped and no data will be transmitted any further.

## 6.2 User Manual for Receiver

User manual provide relevant information for usage of this software.

### 6.2.1 Add Target Sender

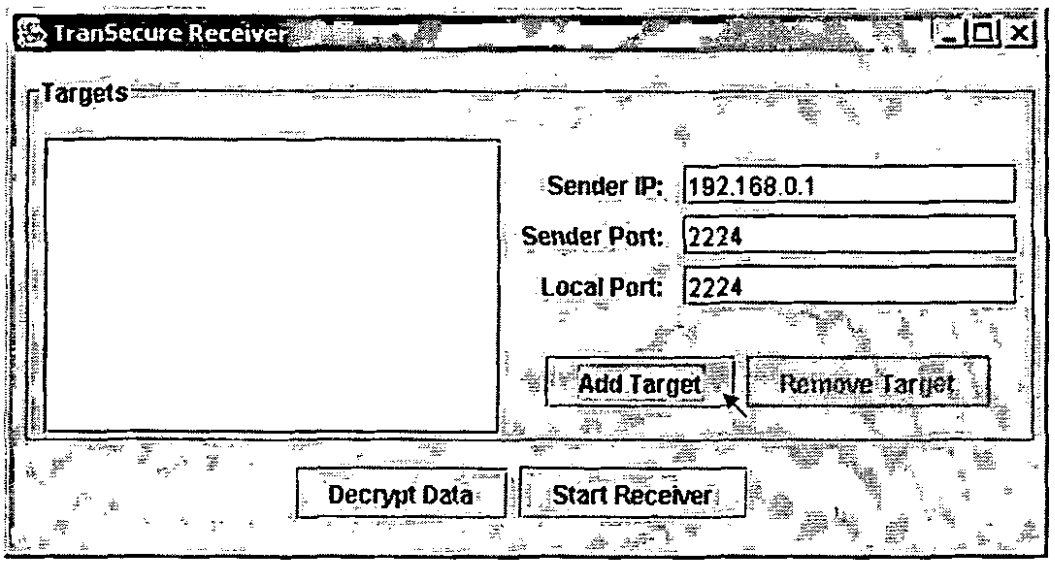


**Figure 6.8: Add Target Sender**

The receiving module adds senders to its list. To add a target sender, the IP address and port of the transmitting terminal along with the local port that is receiving data should be inserted and then the "Add Target" button be pressed. A target transmitter will be added to the list of transmitters as our application supports the multicast reception. If more than one sender is added to the list, each sender will have a session of its own.

The string 2224 <--- 192.168.0.1:2224 tells that the local port 2224 is receiving payload from target client 192.168.0.1 port 2224.

A log of the target transmitter list is stored in a file "Xmit.dat" that reloads the list in the program when restarted.

### 6.2.2 Remove Target Sender

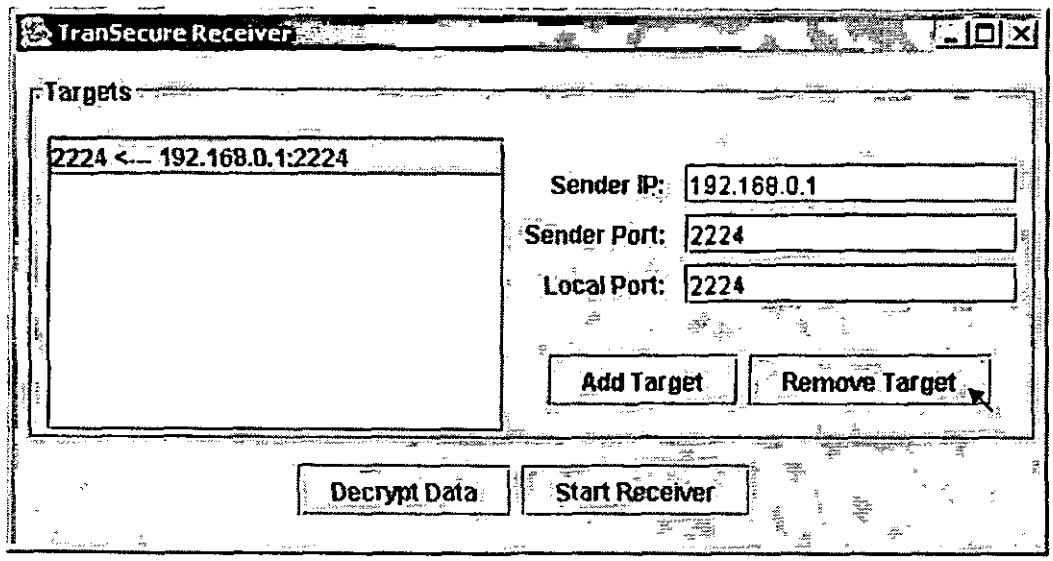


**Figure 6.9: Remove Target Sender**

The receiver module removes target sender from its list. To remove a target sender, an address must be selected and the "Remove Target" is pressed. When the button is clicked it removes the selected address. If any address is not selected, the button "Remove Target" returns nothing and no address is removed. The removed address will also be removed not only from the target list but also for the log file "rx.dat". The status of the application after removing the target address is shown in the following figure.

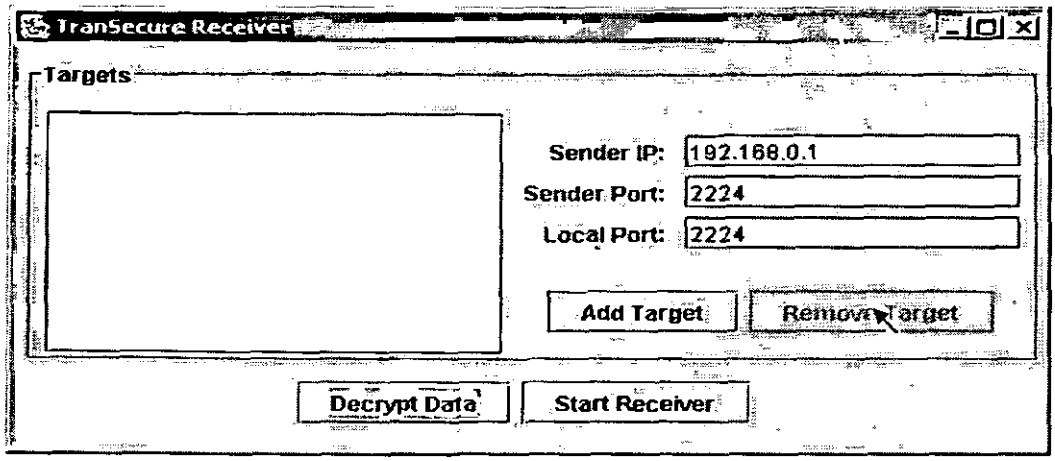


**Figure 6.10: Remove Target**

### 6.2.3 Decrypt Data

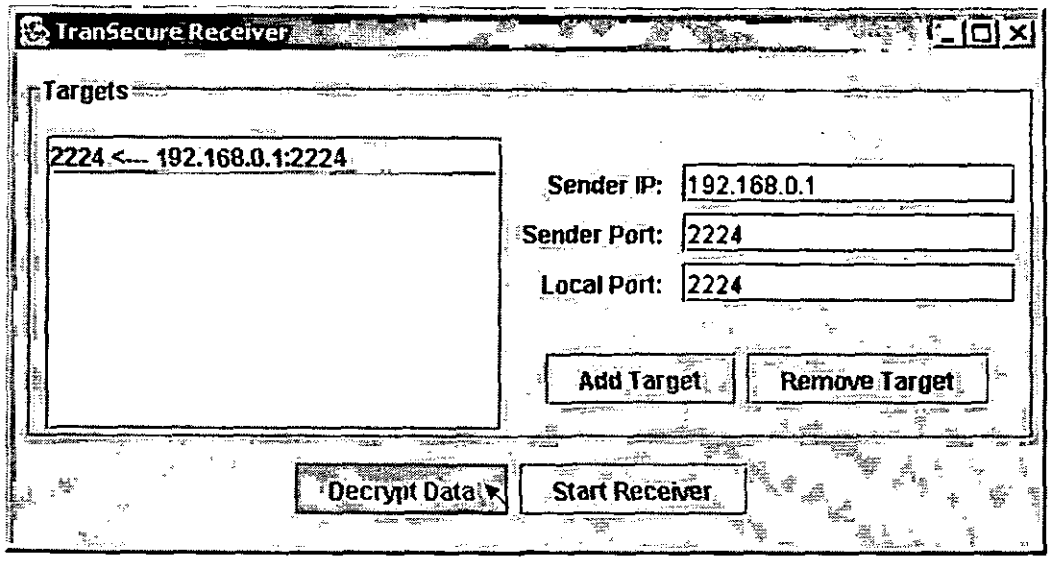


**Figure 6.11: Decrypt Data**

The application provides support for the three encryption algorithms that are proposed by the RFC 3350 which is RFC for Real-time Transport protocol. The three algorithms DES, 3DES and AES are the standard algorithms that can encrypt the payload on the underlying protocol.

When the button "Decrypt Data" is pressed a new socket connection is established and the algorithm name is requested and decryption will be done accordingly.

### 6.2.4 Start Receiver

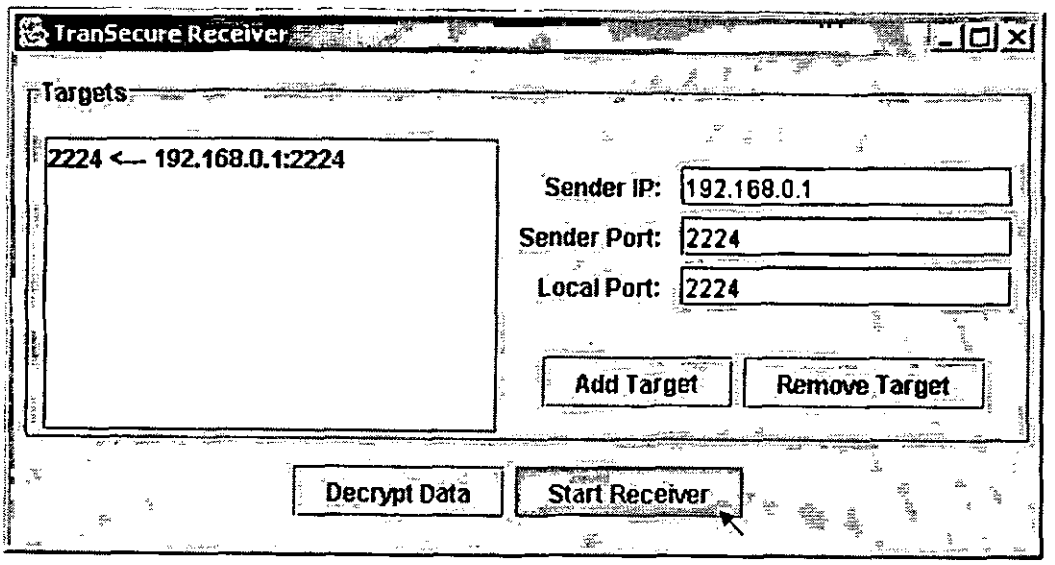


**Figure 6.12: Start Receiver**

After the source addresses are added to the list, the encryption cipher is requested, the system is ready to receive the payload across the network from the listed senders.

The system will receive data from the transmitter whose address is listed. The media stream will test the payload and make as many sessions for each stream. For example, if any file has both audio and video, the each stream will have separate session on separate ports with the receiver.

### 6.2.5 Stop Receiver

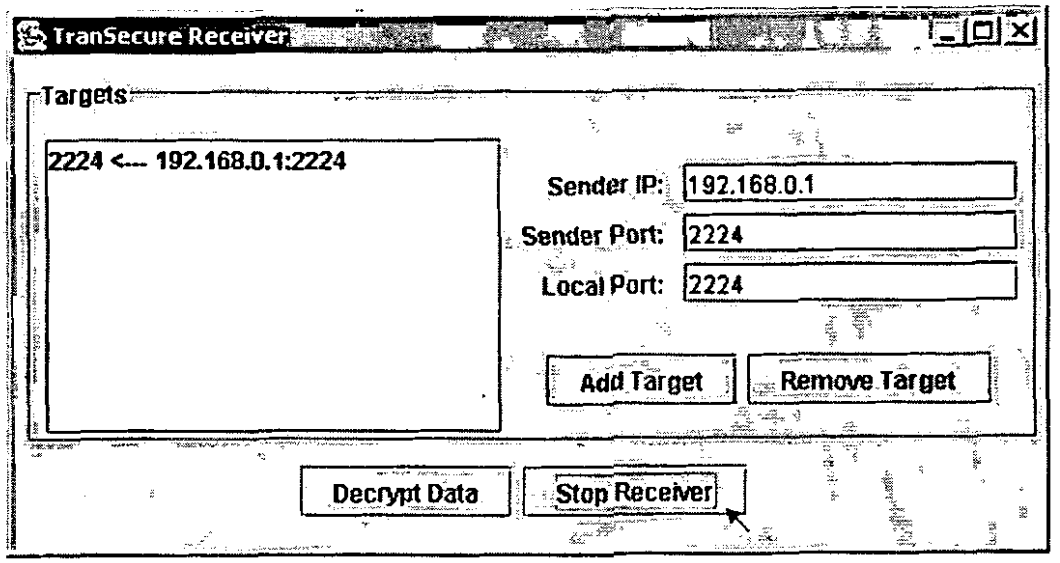


**Figure 6.13: Stop Receiver**

After the successful reception of the data, the application let the user to stop the transmission. The transmission from the sender will be stopped and no data will be received any further.

# Appendices

## Appendix A: Glossary

## Appendix B: References

## Appendix A: Glossary

**AAL5** ATM Adaptation Layer 5. An ATM sub layer for computer networks.

**AH** Authentication Header, and IPsec header that authenticates by cryptographic means datagram's origin and verifies it's integrity.

**ATM** Asynchronous Transfer Mode. A common cell based network technology.

**CBC** Cipher Block Chaining. Encryption mode which hides reoccurring patterns from encrypted stream.

**DES** Data Encryption Standard. Robust cipher which is currently considered somewhat weak due to it's small key size.

**ESP** Encrypted Security Payload, and IPsec header that indicates that datagram is encrypted.

**IPsec** Network level encryption and authentication service for IP networks. IPsec is not widely used, but it is expected to gain popularity in near future.

**IPX** Internetwork Packet Exchange. A legacy network protocol by Novel.

**Mixer** RTP device that changes synchronization source identifiers.

**RTP** Real-time transport Protocol.

**RTCP** RTP Control Protocol.

**RTSP** Real-time Streaming Protocol

**SAP** Session Announcement Protocol.

**SDP** Session Description Protocol.

**SIP** Session Initiation Protocol.

**SNMP** Simple Network Management Protocol

**SSRC** Synchronization source. An unique 32 bit identifier associated with single RTP packet stream originator. SSRC is independent of transport address.

## Appendix B: References


1. **"Security Services for Multimedia Conferencing"**, Stubblebine, S.,16th National Computer Security Conference,(Baltimore, Maryland), pp. 391--395, September 1993.
2. **NIST, "Advanced Encryption Standard (AES)"**, FIPS PUB 197, http://www.nist.gov/aes/
3. Bruce Schneier. **Applied Cryptography: Protocols, Algorithms, and Source Code in C**. John Wiley and Sons, Inc., New York, NY, USA, second edition, 1996.
4. Hellman, M. E., **"A cryptanalytic time-memory trade-off"**, IEEE Transactions on Information Theory, July 1980
5. **RFC 3711 The Secure Real-time Transport Protocol (SRTP)** M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman.
   March 2004.
6. Ville Hallivouri , **"Real-time Transport Protocol Security"** Tik-110.501 Seminar on Network Security HUT TML 2000
7. **RFC 3550 RTP: A Transport Protocol for Real-Time Applications**
   H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. July 2003.
   (Obsoletes RFC1889)
8. **"Voice-over-IP-over-wireless"**, Svanbro, K., Wiorek, J. and B. Olin, Proc. PIMRC 2000, London, Sept. 2000.
9. **RFC 3551, "RTP Profile for Audio and Video Conferences with Minimal Control"**, Schulzrinne, H. and S. Casner, July 2003.
10. **www.java.sun.com**
11. **Merlin Hughes**, Java Network Programming, Manning Publications Co., 2001, 2nd edition.
12. **Deitel & Deitel**, Java How To Program, Prentice Hall, 2002, 4th edition
13. **java.sun.com\products\java-media\jmf\2.1.1\guide**
    Sun Micro System's official JMF Programmers' guide

**INFORMATION TECHNOLOGY JOURNAL**

A Quarterly Publication of ANSInet

Saad Rafique
Department of Computer Sciences, Faculty of Applied Sciences, International Islamic University, Islamabad, Pakistan

04-Aug-2004

Subject: **Galley proof of Article No. 171-PJIT**

Dear Saad Rafique,

We hope you will receive this letter with good health.

Please find enclosed herewith galley proof of above mentioned manuscript for final checking. Please point out the corrections clearly on the manuscript and also provide the typed corrections on a separate sheet also. Your quick response will help us to accommodate your article in the coming issue.

According to our record printing cost is still laying pending. It is therefore, requested to please send printing cost with the checked galley proof.

ANSInet cordially invites to please visit the Journal's website http://www.ansinet.net

Waiting for your quick response.

Regards

Muhammad Imran Pasha
Account Manager

171-PSIT.

Information Technology Journal 3 (): cc cc, 2004
ISSN 1682-6027


# Analysis of Real-time Transport Protocol Security

Junaid Aslam, Saad Rafique and S. Tauseef-ur-Rehman
Department of Computer Sciences, Faculty of Applied Sciences,
International Islamic University, Islamabad, Pakistan

**Abstract:** This study describes RTP and its security related features. The emphasis is on the effect and efficiency of the cryptographic and authentication algorithms recommended by RTP and its profiles. For judging efficiency of the recommend algorithms a java based implementation was developed and a comparison between them was discribed in this study. The study also describes in brief the security features that are available when RTP is used along with higher level protocols like SIP, SAP, SDP and H.323.

**Key words:**

**Introduction to real-time transport protocol (RTP):** Real-time Transport Protocol[1] is an application level protocol that is intended to transmit real-time data such as audio and video. RTP is used for multi-media sessions like business conferences and telemedicine sessions. These sessions support both the unicast and multicast sessions provided the underlying network is multicast capable. It facilitates network transport functions but does not guarantee timely delivery of packet.

RTP is an application level protocol that provides application level framing and is integrated into application processing rather being implemented as a separate protocol stack.

RTP consist of two parts RTP and RTCP, where, RTP is responsible for providing media transmission, while RTCP provides the feedback information on transmission quality.

**Introduction to RTP Profiles:** RTP is never intended to be a complete protocol but as a framework for building application protocols[1] and thus in contrary to other protocols, it is usually implemented by each application according to its requirements for which profiles are defined. A profile defines extensions or modifications to RTP that are specific to a class of applications, services and algorithms that may be offered.

**Secure real-time transport protocol (RTP):** The security features provided by RTP may not fulfill the needs of applications that need extensive security. Such application can consider the use of secure real-time transport protocol (SRTP)[2]. SRTP is a RTP profile that is meant to provide more conventional security services that may be offered. It defines the set of additional cryptographic algorithms and authentication algorithm and allows introducing new ones.

**RTP profile for audio and video conferences with minimal controls:** This profile[3] describes how audio and video data may be carried within RTP and describes the usage of fields left unspecified in RTP. This profile provides a framework for new profiles that may be defined. The SRTP profile is an extension to this profile since all aspects of this profile may apply with addition of SRTP security features.

**Security consideration:** Studies have shown that users may be more sensitive to privacy concerns with audio and video communication than they have been with more traditional forms of network communications[4]. RTP relies on services provided by lower layer protocol for most of its security requirements. However, some methods are described for authentication and some algorithms are specified for attaining confidentiality by RTP or its profiles.

**Confidentiality:** RTP sessions like business conferences and telemedicine sessions do require confidentiality. Confidentiality means that only the intended receiver can decode the received packet; for others, the packet contains no logical information. Confidentiality of the content is attained by encryption. RTP and its profile have recommended few algorithms that may be used for encrypting RTP payload.

**DES:** Data encryption standard (DES) is the default cryptographic algorithm for RTP applications. The mode used is DES-CBC. The DES-CBC mode was chosen

**Corresponding Author:** Saad Rafiq, Department of Computer Sciences, Faculty of Applied Sciences, International Islamic University, Islamabad, Pakistan E-mail:saadrafique@hotmail.com

because it has shown to be easy and practical to use in experimental audio and video communication. The CBC mode has the advantage of having the random access property for decryption which guarantees that any lost packet could only prevent decoding of itself and the following packets of that specific block[5].The remaining transmission is not affected by this loss.

DES is good choice as the overhead caused by it is hardly noticeable compared to the CPU requirements of modern compression algorithm of voice and video. Also, it is a fast to execute algorithm that is good for real time data. But DES has been found to be easily broken with specialized hardware[6]. Also, the DES is mainly designed for hardware implementation. It is difficult to implement efficiently on software and therefore, not an optimal choice for huge amount of time sensitive data.

**Triple DES:** Since DES has been easily broken, the usage of a stronger encrypting algorithm is recommended. The leading candidate for a replacement to DES is triple DES. Triple DES is a three level construct using DES at each level. Unfortunately the triple DES is much slower and takes a lot of processing time since it is encrypted three times over. Also the block size of triple DES is not bigger than DES, the 64 bit block size has security implications of its own. Triple DES fails to address the problems of manipulating individual bits of the following packets if an anacker manages to get a packet.

AES: RTP profile SRTP recommends AES[7] with counter mode (CM) and f-8 mode. The f-8 mode is used for wireless transmission. AES overcomes the flaws of individual bit manipulation, introduced by CBC mode because it has the property that the encryption and decryption of one packet does not depends on preceding packets. The security of CM has been proven by Bellare[8]. Their analysis shows the security of CM can be better than that of CBC.

AES would be a good choice because it has a larger block size of 128 bits. AES offers larger encryption key. The key size can vary from 128,196 and 256 bit and can be used according to user requirements. In addition to increased security that comes with larger key sizes the AES can encrypt data much faster than triple DES, due to the reason that AES has a lot of inherent parallelism in implementation making it easy to utilize processor resources efficiently.

**Efficiency comparison of recommended encryption algorithm:** A Java based implementation of real time audio and video transmission "TranSecure" was developed to provide the encrypted transmission of real time data. The data was encrypted using the recommended algorithms (Fig. 1) and the efficiency was noted for transmitting 1, 383 KB test file using a 1.GHz P-IV machine over a 100 Mbps fast Ethernet.

**Authentication:** Authentication means guarantee of originator and of electronic transmission. It is useful to verify the authentication of the receiver to assess the trustworthiness. RTP does not specify any authentication except that implicit authentication is assumed if encryption is present. However, its profiles specify some authentication methods.

**MD5:** This RTP profile[3] specifies hashing algorithm MD5 and describes a method that retrieves authentication key from the password that may be used for authentication[9]. The massage digest MD5 hash algorithm is considered to be as strong as 128 bit hash code so the difficulty of finding a message with a given digest is of the order $2^{128}$ operations.

**Secure hash algorithm:** The profile SRTP[2] recommends that each SRTP stream should be protected by SHA-1. The default session authentication key-length is 160-bits while the default authentication tag length shall be 80 bits. SRTP recommends that it should not be used without message authentication because the predefined encryption algorithms does not provide any message authentication[2]. The SRTP allows small authentication tags of 32 bits for 3G networks where under certain link technologies, even few additional bytes could result in significant result of efficiency[10]. The security provided by 32 bit tag is limited and is allowed for restricted set of applications.

**Comparison of MD5 and SHA-1:** 160-bit SHA-1 is 32 bit longer than 128-bit MD5, thus if neither algorithms contain any structural flaws that are vulnerable to cryptanalytic attack, then SHA-1 is a stronger algorithm.

The SHA involves more steps (80 vs 64) and must process 160 bit buffer compared to 128 bit buffer of MD5. Thus SHA should execute about 25 times slower than MD5 on same hardware.

A comparison of both the authentication tags in the Java based implementation is given in Fig. 2.

**Security provided by underlying protocols:** RTP itself relies either on underlying protocols for security purposes or the higher level protocols that are implemented on RTP framework, define its own security features that may be used. Below here we will discuss the protocols and the security that can be provided by them when used alongwith RTP.

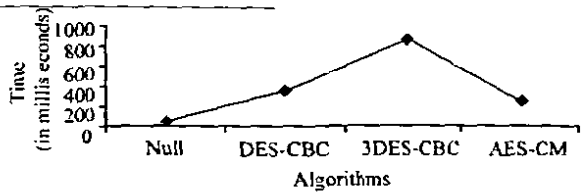


Fig. 1: Comparison of time taken by recommended algorithms

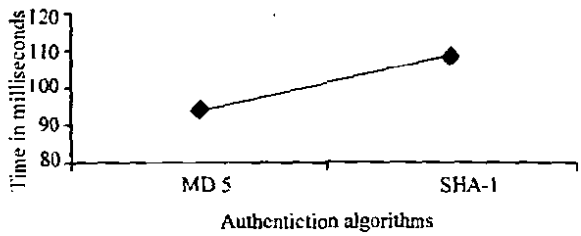


Fig. 2: Comparison of recommended authentication algorithms

**IPSec:** IPSec[11] is considered to be responsible for providing security features, as being the underlying protocol of RTP. It provides services for confidentiality and for authentication it uses its authentication header. IPSec is a comprehensive solution for unicast sessions but it has failed to specify the support for multicast sessions. Thus the higher level protocols need to define own security services.

**Session initiation protocol:** This protocol has the particular ability of distributing the encryption keys and other security parameters. It supports various encryption algorithms and methods. That can provide security to RTP when used with it.

**Session announcement protocol:** This protocol is intended for broadcast information. It has variety of authentication methods such as PGP. Since it is intended for public session announcement, encryption is discouraged.

**Session description protocol:** This protocol may be considered as the high level protocol than SIP and SAP and consequently it usually works in conjunction with SIP and SAP to transport itself. Keys for RTP are distributed using SDP. SDP itself is neither encrypted nor authenticated and usually uses SIP and SAP for such services.

**H. 323:** This is the most widely used protocol for applications such as telephony and VOIP. It has profile like RTP that describes security services (Table 1). The possible security services have been defined by [X4] which describes that H.323 be used with SRTP instead and hence provides security services that also supports multicast sessions.

Table 1: Common RTP configuration

| | RTP | RTP+IPSec | RTP+SDP | SRTP+H.323 |
|---|---|---|---|---|
| Key/algorithm setup | No | Yes | Yes | Yes |
| Confidentiality | Yes | Yes | Yes | Yes |
| Session authentication | Implicit | Yes | Yes | Yes |
| Session integrity | Some | Yes | Some | Yes |
| Transmitter authentication | Unicast | Unicast | Unicast | Multicast |
| Multicast support | Yes | No | Yes | Yes |

**Conclusion:** Security facilities provided by the RTP protocol alone are inadequate. RTP has an excellent design since it provides a framework for high level protocol which in turn can implement its own security services that may eventually provide benefit to RTP itself. RTP cannot rely on underlying network just because it is transmitted over IP and considering IPSec will consequently provide the security services. Services provided by IPSec are not useful for protocol other than IP and also it doesn't support multicast sessions. RTP is network independent and could use the other protocols like ATM/AAL5 for transmission of Real-time data for which IPSec won't be viable. Services provided by RTP are quite comprehensive though it still has failed to provide adequate authentication for wireless networks and the key distribution mechanism. Authentication with SHA may be useful but the key size may be burdensome for wireless networks.

RTP is a flexible protocol and allows new encoding techniques that may provide secure communication that may solve these problems. RTP in conjunction with other protocols can also be used for security purposes that may eventually solve most of the problems.

## REFERENCES

1. Schulzrinne, H., S. Casner, R. Frederick and V. Jacobson, 2003. RTP: A transport protocol for real-time applications, RFC3550, July 2003.
2. Baugher, M., R. Blaom, E. Carrara, D. McGrew, M. Naslund, K. Norman and D. Oran. 2004. Secure Real-time Transport Protocol: RFC 3711, March 2004.
3. Schulzrinne, H. and S. Casner, 2003. RTP profile for audio and video conferences with minimal control: RFC 3551, July 2003.
4. Stubblebine, S., 1993. Security services for multimedia conferencing. In 16th National Computer Security Conference, (Baltimore, Maryland), pp: 391-395. September, 1993.

5. Bruce, S., 1996. Applied Cryptography: Protocols, Algorithms and Source Code in C. John Wiley and Sons, Inc., New York, NY, USA, 2nd Edn., 1996.
6. Electronic Frontier Foundation. Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design. O'Reilly and Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA.
7. NIST, Advanced Encryption Standard (AES), FIPS PUB 197, http://www.nist.gov/aes/ **(date of list visit)**
8. Bellare, M., A. Desai, E. JokiPii and P. Rogaway, 1997. A concrete security treatment of symmetric encryption:analysis of the DES modes of operation. Proceedings 38th Symp. Found. Computer Science, IEEE, 1997. A revised version is available online at http://www-cse.ucsd.edu/users/mihir.
9. Ville, H., 2004. Real-time transport protocol security, Tik-110.501 Seminar on Network Security HUT TML 2000.
10. Svanbro, K., J. Wiorek and B. Olin, 2000. Voice-over-IP-over-wireless. Proc. PIMRC 2000, London, Sept. 2000.
11. Kent, S. and R. Atkinson, 1998. Security architecture for the internet protocol: RFC 2401, November 1998.