Design of Secure Transmission of Multimedia Data Using SRTP on Linux Platform

Shashidhar H.G., Sanket Dessai, Shilpa Chaudhari
Department of Computer Engineering, M.S. Ramaiah School of Advanced Studies, Bangalore-560058, India

Article Info

Article history:

Received Dec 2, 2014
Revised Mar 18, 2015
Accepted Apr 15, 2015

Keyword:

Authentication
Security
MPEG
RTP/RTCP
SRTP

ABSTRACT

This paper aims for providing a viable solution for security in streaming media technology. Service providers do not want the end users to capture and duplicate streaming media data. Once captured data can be re-distributed to millions without any control from the source. Licensing issues also dictate the number of times end user may utilize the data. Encryption is not sufficient as it leaves the system vulnerable to duplication and recording after decryption. In this paper an attempt has been made to transmit digital multimedia data to multiple users. The transmission of the video/audio data has been attempted from one PC to another PC. While doing this, security considerations have to be taken care by using suitable encryption/decryption techniques. A research carried out on the different data transmission protocols reveals that the Secure Real Time Transport Protocol (SRTP) is one of the best available protocols. Hence the SRTP has been deployed in this project on Linux OS using socket programming. The code for the transmitter and the receiver is designed and developed around the SRTP library for transmission of multimedia data. The solution is illustrated by choosing an example of a video clip for transmission and reception. This model increasing the security of streaming media and adds a measure of integrity protection, but it is primarily intended to aid in replay preventions.

Corresponding Author:

Sanket Dessai,
Departement of Computer Engineering, M.S. Ramaiah School of Advanced Studies,
#470-P, Peenya Industrial Area, Bangalore, Karnataka, India. 560058.
Email: sanketdessai@msrsas.org

1. INTRODUCTION

Due to recent advances in technology, streaming media including real time audio and video conferencing has become a popular and important aspect of telecommunication. Often, streaming content providers do not want the end users to capture and duplicate the streamed media, since once such data is captured it can be freely re-distributed without any control from the source. Privacy and integrity are crucial for successful video or audio conferencing. Note that data encryption [1] does not generally suffice, since it leaves the system vulnerable to duplication and recording after decryption. Possible sources of streamed media include stored data, live broadcasts and interactive conferencing. We focuses on live broadcast media as our case study but the results are easily extended to stored media or interactive conferencing.

Secure real-time communication over insecure networks generally involves two major security concern authentication and privacy [2]. The real-time transport protocol (RTP) is used to transport a media stream between two multimedia terminals. Secure RTP employs encryption techniques on the RTP stream [3].

2. SRTP (SECURE REAL-TIME TRANSPORT PROTOCOL)

The Secure Real-time Transport Protocol (SRTP) defines a framework which provides confidentiality, message authentication, and replay protection for both unicast and multicast RTP and RTPCP streams. SRTP is very suitable for VoIP applications, especially those involving low-bit rate voice codecs (i.e.G.729, iLBC, MELP, etc.) since it can be used with header compression and has no significant impact on Quality of Service. It can also be used to with JPEG, MPEG2, and MPEG4 to securely stream video in multimedia applications. SRTP can achieve high throughput and low packet expansion even in environments that are a mixture of wired and wireless networks. [8]

SRTP is the security layer which resides between the RTP/RTCP application layer and the transport layer, generating SRTP packets from the RTP/RTCP stream and forwarding these to the receiver. Similarly, it also transforms incoming SRTP packets to RTP/RTCP packets and passes these up the stack.

The cryptographic state information associated with each SRTP stream is termed the cryptographic context. It must be maintained by both the sender and receiver of SRTP streams. If there are several SRTP streams present within a given RTP session, separate cryptographic contexts must be maintained for each. A cryptographic context includes any session key (a key directly in encryption/message authentication) and the master key as well as other working session parameters. While SRTP does not define a precise mechanism to implement key exchange, it does provide for several features which make key management easier and heighten overall key security. The single master key is used to provide keying material for a key derivation function. This can generate the initial session keys, as well as provide new session keys periodically to ensure that there will be a limited amount of cipher text produced by any given encryption key. Salting keys are used to provide protection against various assaults such as pre-computation and time-memory attacks. [8]

3. REQUIREMENT ANALYSIS

- **The network protocol used to transmit the multimedia data:** Secure Real Time protocol (SRTP) is used to transmit multimedia over the network. The designers of SRTP focused on developing a protocol that can provide adequate protection for media streams but also maintain key properties to support wired and wireless networks in which bandwidth or underlying transport limitations may exist.
- **Securing the multimedia data:** Private Key cryptography is chosen. The algorithm for encryption and decryption chosen is AES. Here selective encryption/decryption method has been chosen as opposed to conventional heavy weight encryption as the computation time is reduced.
- **MPEG1** has been chosen as the type of multimedia data in this project.

4. DESIGN: CLIENT SERVER DESIGN

Applications are usually designed so that one computer acts as a server, providing a service to other computers on a network. To access a server, a program is run on a user's computer this is called a client program. The program establishes a connection through the network allowing communication with the server. The security model includes a license manager to manage access to requested data. The operation of this feature will be described in more detail below.

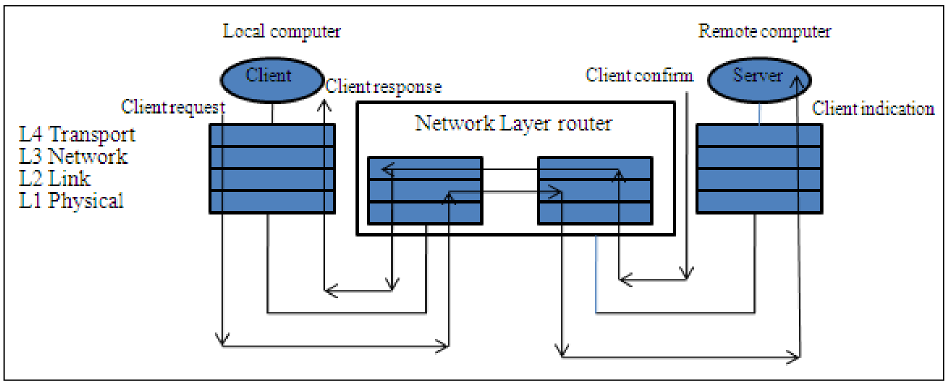


Figure 1. Client/Server Interaction across a Packet Data Network

Another security feature involves a scrambling algorithm, which is employed by the server and a corresponding de-scrambling algorithm, which is employed by the client. A scrambling algorithm should be unknown to a potential attacker and an attacker must be required to break the scrambling algorithm in order to recover any of the data. In addition, the server must have access to a significant number of distinct scrambling algorithms.

Scrambling serves two purposes. First, the scrambling algorithm creates a layer of obfuscation, making reverse engineering of the client software more difficult. Second, scrambling provides for a high degree of individualization of the client software. Consequently, scrambling algorithms that are unknown to a potential attacker are preferred.

Perhaps the ideal scrambling algorithm is a cryptosystem, since it could be applied to all of the data. However, no cryptographic algorithm is considered secure until it has undergone extensive peer review and withstood the test of time. But the scrambling algorithm is not essential for cryptographic strength, since standard strong encryption algorithms are employed for cryptographic strength. Given such a set of scrambling algorithms, each client will be equipped with a subset of the available scrambling algorithms. The list of scrambling algorithms known to the client will be encrypted with a key known only to the server, and stored on the client. After authenticating the server, this encrypted list will be passed from the client to the server. When the server receives the list, the server decrypts it and randomly chooses from among the client’s scrambling algorithms. The ID number of the selected scrambling algorithm is then passed from the server to the client. Note that this process eliminates the need for a database containing the mappings between clients and scrambling algorithms.

By having different scrambling algorithms embedded within different clients, and by selecting at random from a client’s algorithms, each client is unique, and each communication between client and server depends not only on different keys, but also on different algorithms embedded in the client software. An attacker, who is able to break one particular piece of content, will likely still have a challenging task when trying to break another piece of content destined for the same client. And even if an attacker completely does reverse engineering on one client, it is likely that he will still need to expend roughly the same effort to attack any other client.

On the server side, the data is scrambled, and then encrypted. On the client side, the data is decrypted and the resulting scrambled data is passed to the media application. The media application passes the scrambled data to the secure device driver (discussed in more detail below), which de-scrambles the data. In this way, the data is obfuscated until the last possible point in the process.

Given the security features, the secure streaming media process proceeds as follows:

The secure web server offers streaming media services.

- A client requests a media file from the server
- The secure authenticates the user and the user authenticates the server
- The license manager verifies that the user on that particular machine is allowed access to the requested media file.
- If the user is allowed access, the License Manger generates two random keys. The first key will be for secure RTP packet encryption using AES and the second key will be the scrambling key used on message blocks of media data.
- The server generates a random number to select from among the scrambling algorithms supported by the client. It generates another random number to be used as the key for the scrambling algorithm. Both of these are encrypted (but not scrambled) and passed to the client. The client must acknowledge receipt of this information.
- The server use cipher block chaining (CBC) to scramble the data per packet, with a randomly selected initialization vector (IV) included with each packet for cryptographic terminology and information.
- The secure RTP algorithm with the Advanced Encryption Algorithm (AES) with 128-bit key is applied to the scrambled data in each packet. The packets are CBC encrypted with a random IV included in each.
- The scrambled and encrypted secure RTP packets are transmitted over the network.
- The client opens the secure media application from the file. Block diagram of this is shown in figure 1.
- The media application requests the secure RTP decryption key. The user must authenticate in order for the client to obtain the decryption key.
- The media application strips the secure RTP header and sequences the packets.
- The media application is oblivious to the scrambling. It therefore writes the scrambled data to the device that plays the file. The interaction between client and server is shown in figure 2.

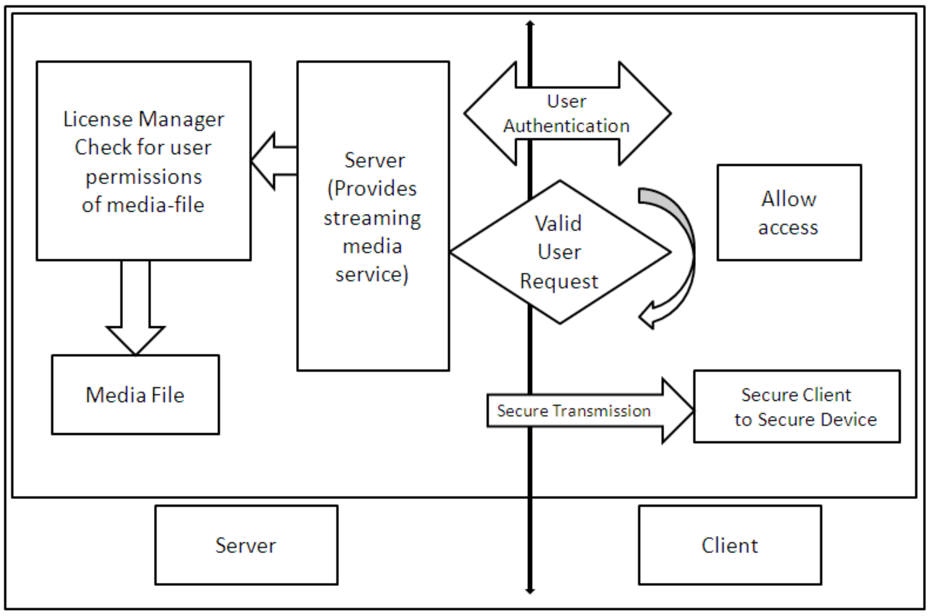
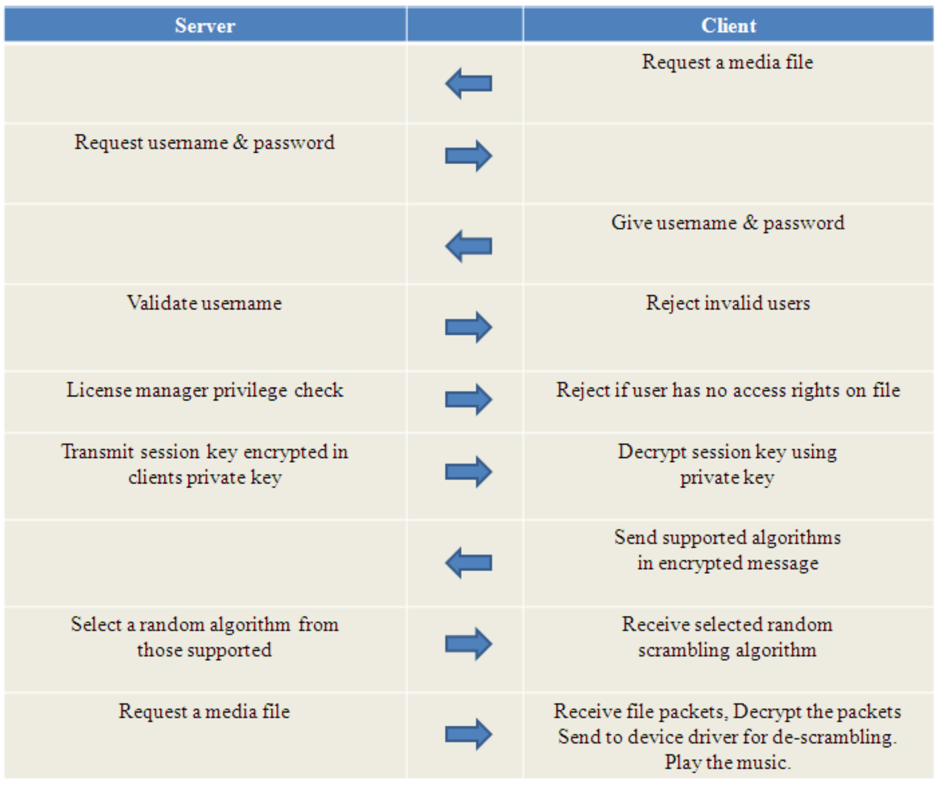

License Manager Check for user permissions of media-file
Media File
Server (Provides streaming media service)
User Authentication
Valid User Request
Allow access
Secure Transmission
Secure Client to Secure Device
Server
Client

Figure 2. Block Diagram of the Design of the Secure System with Client/Server Interaction



Server		Client
	⬅	Request a media file
Request username & password	➡	
	⬅	Give username & password
Validate username	➡	Reject invalid users
License manager privilege check	➡	Reject if user has no access rights on file
Transmit session key encrypted in clients private key	➡	Decrypt session key using private key
	⬅	Send supported algorithms in encrypted message
Select a random algorithm from those supported	➡	Receive selected random scrambling algorithm
Request a media file	➡	Receive file packets, Decrypt the packets Send to device driver for de-scrambling. Play the music.

Figure 3. Flow Diagram for the Secure SRTP Transmission

5. IMPLEMENTATION: CLIENT SERVER DESIGN

The implementation section describes in detail the various components of the secure streaming media system. For a complete source code listing kindly see the appendix. Some of the functions used are listed below.

- err_status_t srtp_init (void)
  srtp_init () initializes the srtp library.
- err_status_t srtp_protect (srtp_t ctx, void *rtp_hdr, int *len_ptr)
  srtp_protect () is the Secure RTP sender-side packet processing function.
- err_status_t srtp_unprotect (srtp_t ctx, void *srtp_hdr, int *len_ptr)
  srtp_unprotect () is the Secure RTP receiver-side packet processing function.
- err_status_t srtp_create (srtp_t *session, const srtp_policy_t *policy)
  srtp_create () allocates and initializes an SRTP session.
- err_status_t srtp_add_stream (srtp_t session, const srtp_policy_t *policy)
  srtp_add_stream () allocates and initializes an SRTP stream within a given SRTP session.
- err_status_t srtp_remove_stream (srtp_t session, uint32_t ssrc)
  srtp_remove_stream () deallocates an SRTP stream.
- void crypto_policy_set_rtp_default (crypto_policy_t *p)
  crypto_policy_set_rtp_default () sets a crypto policy structure to the SRTP default policy for RTP protection.
- void crypto_policy_set_rtcp_default (crypto_policy_t *p)
  crypto_policy_set_rtcp_default () sets a crypto policy structure to the SRTP default policy for RTCP protection.
- void crypto_policy_set_aes_cm_128_hmac_sha1_32 (crypto_policy_t*p)
  crypto_policy_set_aes_cm_128_hmac_sha1_32 () sets a crypto policy structure to a short-authentication tag policy
- void crypto_policy_set_aes_cm_128_null_auth (crypto_policy_t *p)
  crypto_policy_set_aes_cm_128_null_auth () sets a crypto policy structure to an encryption-only policy
- void crypto_policy_set_null_cipher_hmac_sha1_80 (crypto_policy_t *p)
  crypto_policy_set_null_cipher_hmac_sha1_80 () sets a crypto policy structure to an authentication-only policy
- err_status_t srtp_dealloc (srtp_t s)
  srtp_dealloc () deallocates storage for an SRTP session context.
- err_status_t crypto_get_random (unsigned char *buffer, unsigned int length)
  writes a random octet string.

LibSRTP: To install libSRTP, download the latest release of the distribution from **srtp.sourceforge.net**. The format of the names of the distributions is srtp-A.B.C.tgz, where A is the version number, B is the major release number, C is the minor release number, and tgz is the file extension. Unpack the distribution and extract the source files the directory into which the source files will go is named srtp.Implementation details in Linux using kernel 2.4.16 are given below in detail.
Server components: libSRTP was installed on the server and uses the GNU autoconf and make utilities. In the srtp directory, run the configure script and then make:

```
. /configure [options]
make
```

The configure script accepts the following options:
- Help-provides a usage summary.
- Disable-debug compiles libSRTP without the runtime dynamic debugging system.
- Enable-generic-aesicm compile in changes for ismacryp
- Enable-syslog use syslog for error reporting.
- Disable-stdout disables stdout for error reporting.
- Enable-console use /dev/console for error reporting
- Gdoi- use GDOI key management (disabled at present).

License Manager:
- The receiver sends a string of supported algorithms in encrypted form. The server randomly elects one of the supported algorithms, maps the client algorithm to its server algorithm and sends the selected algorithm number to the receiver. The server then starts sending the streaming data in predefined packet size.
- The license manager maintains a list of multimedia data files and corresponding username and number of times the user is permitted to invoke that file. On each invocation, the license manager decrements by one the allowed number for that particular user. If the user is allowed infinite number of accesses, then the license manager will not decrement the number of times allowed on each execution. Access is

allowed on a particular file only if the times allowed are greater than zero. In practice this logic could be easily implemented using a secure database system.
- The license manager also maintains a list of broken scrambling algorithms. If the license manager detects that all the supported algorithms at the receiver end are broken, it will ask the server to terminate its connection with the receiver without giving any explanation to the receiver.

Receiver Side Components
- libSRTP was installed on the receiver side and uses the GNU autoconf and make utilities as we did in server side , same as to be carried out in receiver side .
- The usage for client_server.c is
- client_server [[-d <debug>]* [-k <key> [-a][-e]] [-r] dest_ip dest_port][-l]

5.1. SRTP Crytographic Context

Each SRTP stream requires the sender and receiver to maintain cryptographic state information. This information is called the "cryptographic context".
SRTP uses two types of keys:
1. Session keys and
2. Master keys.

By a "session key", we mean a key, which is used directly in a cryptographic transform (e.g., encryption or message authentication), and by a "master key", we mean a random bit string (given by the key management protocol) from which session keys are derived in a cryptographically secure way. The master key(s) and other parameters in the cryptographic context are provided by key management mechanisms external to SRTP. [15]

5.2. Mapping SRTP Packets to Cryptographic Contexts

RTP session for each participant is defined [RFC3550] by a pair of destination transport addresses (one network address plus a port pair for RTP and RTCP), and that a multimedia session is defined as a collection of RTP sessions. For example, a particular multimedia session could include an audio RTP session, a video RTP session, and a text RTP session.
A cryptographic context SHALL be uniquely identified by the triplet context identifier:
context id = <SSRC, destination network address, destination transport port number>

Where, the destination network address and the destination transport port are the ones in the SRTP packet. It is assumed that, when presented with this information, the key management returns a context with the information.

As noted above, SRTP and SRTCP by default share the bulk of the parameters in the cryptographic context. Thus, retrieving the crypto context parameters for an SRTCP stream in practice may imply a binding to the correspondent SRTP crypto context. It is up to the implementation to assure such binding, since the RTCP port may not be directly deducible from the RTP port only. Alternatively, the key management may choose to provide separate SRTP- and SRTCP- contexts, duplicating the common parameters (such as master key(s)). The latter approach then also enables SRTP and SRTCP to use, e.g., distinct transforms, if so desired. Similar considerations arise when multiple SRTP streams, forming part of one single RTP session, share keys and other parameters.

If no valid context can be found for a packet corresponding to a certain context identifier, that packet MUST be discarded. [15]

5.3. Algorithm of SRTP Packet Processing

The following applies to SRTP.Assuming initialization of the cryptographic context has taken place via key management; the sender SHALL do the following to construct an SRTP packet:
Step-1: Determine which cryptographic context to use.
Step-2: Determine the index of the SRTP packet using the rollover counter, the highest sequence number in the cryptographic context, and the sequence number in the RTP packet.
Step-3: Determine the master key and master salt. This is done using the index determined in the previous step or the current MKI in the cryptographic context.
Step-4: Determine the session keys and session salt key (if they are used by the transform), using master key, master salt, key_derivation_rate, and session key-lengths in the cryptographic context with the index.
Step-5: Encrypt the RTP payload to produce the Encrypted Portion of the packet, This step uses the encryption algorithm indicated in the cryptographic context, the session encryption key and the session salt (if used) found in Step 4 together with the index found in Step 2.
Step-6: If the MKI indicator is set to one, append the MKI to the packet.

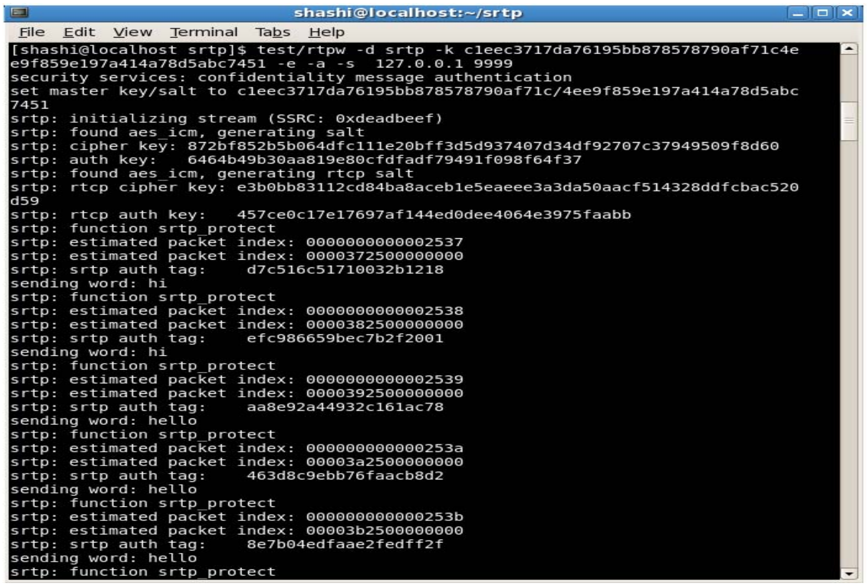
Step-7: For message authentication, compute the authentication tag for the Authenticated Portion of the packet, this step uses the current rollover counter, the authentication algorithm indicated in the cryptographic context, and the session authentication key found in Step 4. Append the authentication tag to the packet.

5.4. Algorithm of Authenticate and Decrypt an SRTP Packet

Step-1: Determine which cryptographic context to us.
Step-2: Run the algorithm to get the index of the SRTP packet. The algorithm uses the rollover counter and highest sequence number in the cryptographic context with the sequence number in the SRTP packet.
Step-3: Determine the master key and master salt. If the MKI indicator in the context is set to one, use the MKI in the SRTP packet, otherwise use the index from the previous step.
Step-4: Determine the session keys, and session salt (if used by the transform) using master key, master salt, key_derivation_rate and session key-lengths in the cryptographic context with the index, determined in Steps 2 and 3.
Step-5: For message authentication and replay protection, first check if the packet has been replayed, using the Replay List and the index as determined in Step 2. If the packet is judged to be replayed, then the packet MUST be discarded, and the event SHOULD be logged. Next, perform verification of the authentication tag, using the rollover counter from Step 2, the authentication algorithm indicated in the cryptographic context, and the session authentication key from Step 4. If the result is "AUTHENTICATION FAILURE", the packet MUST be discarded from further processing and the event SHOULD be logged.
Step-6: Decrypt the Encrypted Portion of the packet, using the decryption algorithm indicated in the cryptographic context, the session encryption key and salt (if used) found in Step 4 with the index from Step 2.
Step-7: Update the rollover counter and highest sequence number, s_l, in the cryptographic context, using the packet index estimated in Step 2. If replay protection is provided, also update the Replay List.
Step-8: When present, remove the MKI and authentication tag fields from the packet.

6. RESULTS

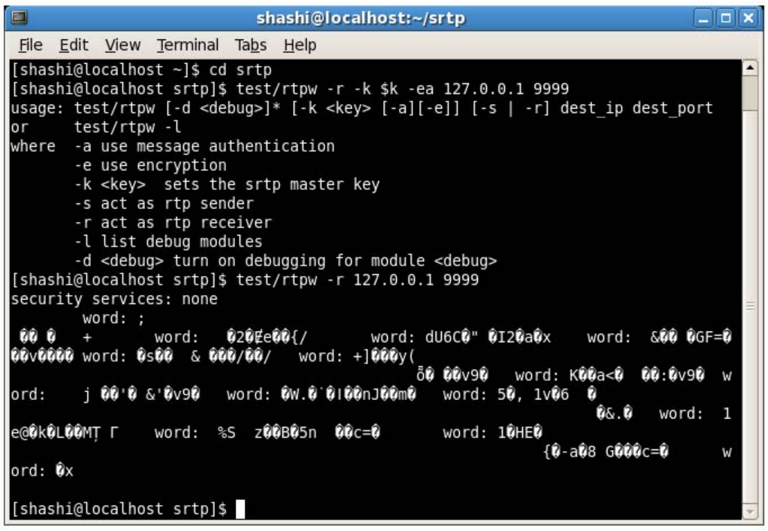
The below given Figure 4, showing the result of sender side (Server), where streaming of words taking place towards client side taking place. This snap shot include the SSRC valve cipher key, authentication key, estimated packet index SRTP authentication tag including master key/salt key given during initial setup. The below given Figure 5, showing the result at client side, where the words sent from server side is in encrypted form from this we can tell that no one can understand anything if that data is accessed in middle during transmission. The below given Figure 6, showing the result at client side, where the words sent from server side is in original form, this is possible after decrypting the data sent by sender.



```
[shashi@localhost srtp]$ test/rtpw -d srtp -k c1eec3717da76195bb878578790af71c4e
e9f859e197a414a78d5abc7451 -e -a -s  127.0.0.1 9999
security services: confidentiality message authentication
set master key/salt to c1eec3717da76195bb878578790af71c/4ee9f859e197a414a78d5abc
7451
srtp: initializing stream (SSRC: 0xdeadbeef)
srtp: found aes_icm, generating salt
srtp: cipher key: 872bf852b5b064dfc111e20bff3d5d937407d34df92707c37949509f8d60
srtp: auth key:   6464b49b30aa819e80cfdfadf79491f098f64f37
srtp: found aes_icm, generating rtcp salt
srtp: rtcp cipher key: e3b0bb83112cd84ba8aceb1e5eaeee3a3da50aacf514328ddfcbac520
d59
srtp: rtcp auth key:   457ce0c17e17697af144ed0dee4064e3975faabb
srtp: function srtp_protect
srtp: estimated packet index: 0000000000002537
srtp: estimated packet index: 0000372500000000
srtp: srtp auth tag:    d7c516c51710032b1218
sending word: hi
srtp: function srtp_protect
srtp: estimated packet index: 0000000000002538
srtp: estimated packet index: 0000382500000000
srtp: srtp auth tag:    efc986659bec7b2f2001
sending word: hi
srtp: function srtp_protect
srtp: estimated packet index: 0000000000002539
srtp: estimated packet index: 0000392500000000
srtp: srtp auth tag:    aa8e92a44932c161ac78
sending word: hello
srtp: function srtp_protect
srtp: estimated packet index: 000000000000253a
srtp: estimated packet index: 00003a2500000000
srtp: srtp auth tag:    463d8c9ebb76faacb8d2
sending word: hello
srtp: function srtp_protect
srtp: estimated packet index: 000000000000253b
srtp: estimated packet index: 00003b2500000000
srtp: srtp auth tag:    8e7b04edfaae2fedff2f
sending word: hello
srtp: function srtp_protect
```

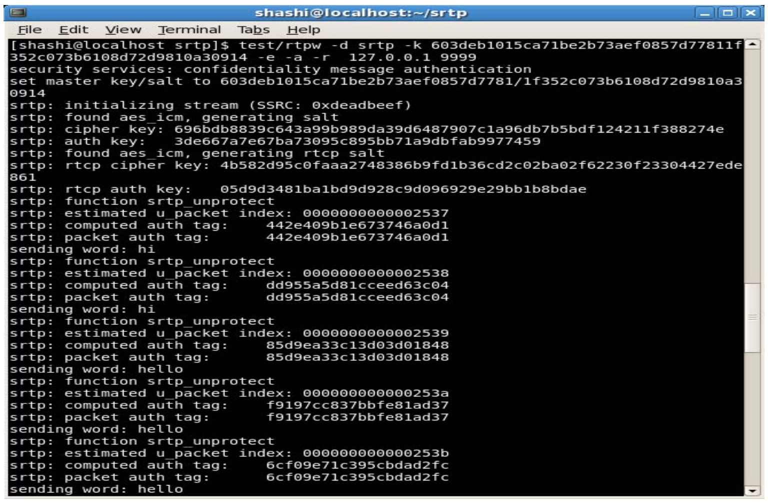
Figure 4. MPEG Video Clip Data Sending From Server to Client

The above given Figure 7, showing the result of sender side, where streaming of MPEG clip taking place towards client side taking place. The below given Figure 8, showing the frame of playing original clip which has to be sent from server side to client side. The below given Figure 9, showing the frame of playing encrypted clip which has sent from server side to client side.The below given Figure 10, showing the frame of playing clip after decrypting the encrypted clip sent from server side to client side.



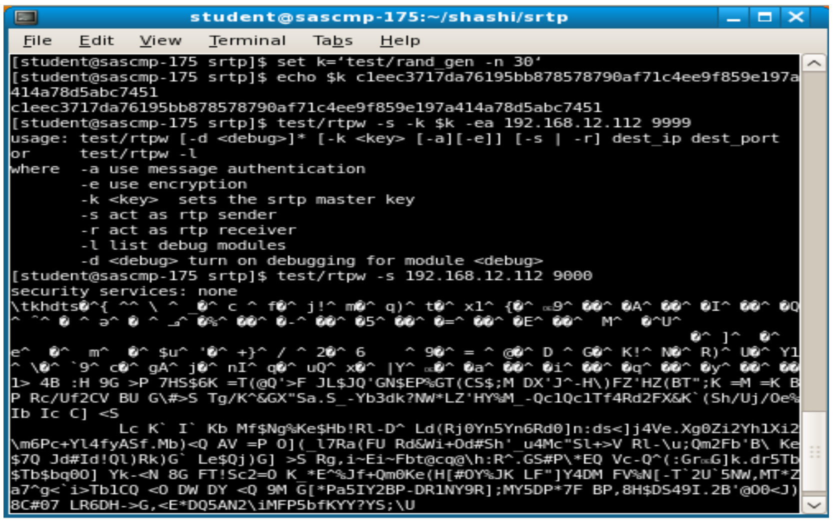
```
[shashi@localhost ~]$ cd srtp
[shashi@localhost srtp]$ test/rtpw -r -k $k -ea 127.0.0.1 9999
usage: test/rtpw [-d <debug>]* [-k <key> [-a][-e]] [-s | -r] dest_ip dest_port
or     test/rtpw -l
where  -a use message authentication
       -e use encryption
       -k <key>  sets the srtp master key
       -s act as rtp sender
       -r act as rtp receiver
       -l list debug modules
       -d <debug> turn on debugging for module <debug>
[shashi@localhost srtp]$ test/rtpw -r 127.0.0.1 9999
security services: none
        word: ;
 �� �    +       word:   �2�Ɇe��{/       word: dU6C�" �I2�a�x    word:  &�� �GF=�
��v���� word: �s��  & ���/��/   word: +]���y(
                                                  õ� ��v9�   word: K��a<�  ��:�v9�  w
ord:    j ��'� &'�v9�   word: �W.�˙�I��nJ��m�   word: 5�, 1v�6  �
                                                                    �&.�   word:  1
e@�k�L��MȚ Γ    word:  %S  z��B�5n  ��c=�       word: 1�HE�
                                                              {�-a�8 G���c=�      w
ord: �x

[shashi@localhost srtp]$ █
```

Figure 5. The Results of Client Side, Where Stream of Words sent from Server in Encrypted Form



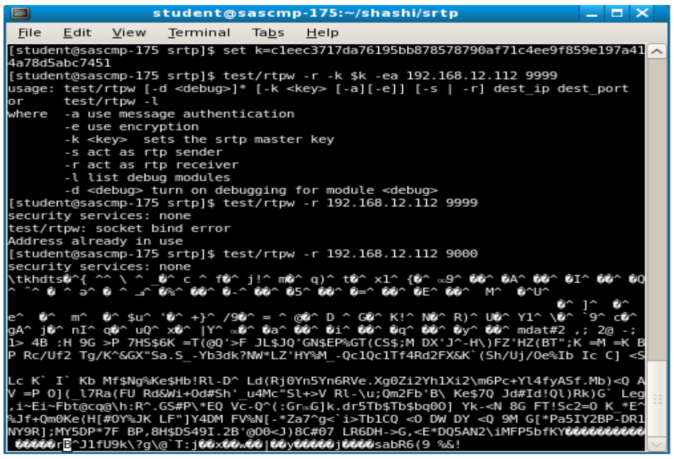
```
[shashi@localhost srtp]$ test/rtpw -d srtp -k 603deb1015ca71be2b73aef0857d77811f
352c073b6108d72d9810a30914 -e -a -r  127.0.0.1 9999
security services: confidentiality message authentication
set master key/salt to 603deb1015ca71be2b73aef0857d7781/1f352c073b6108d72d9810a3
0914
srtp: initializing stream (SSRC: 0xdeadbeef)
srtp: found aes_icm, generating salt
srtp: cipher key: 696bdb8839c643a99b989da39d6487907c1a96db7b5bdf124211f388274e
srtp: auth key:   3de667a7e67ba73095c895bb71a9dbfab9977459
srtp: found aes_icm, generating rtcp salt
srtp: rtcp cipher key: 4b582d95c0faaa2748386b9fd1b36cd2c02ba02f62230f23304427ede
861
srtp: rtcp auth key:   05d9d3481ba1bd9d928c9d096929e29bb1b8bdae
srtp: function srtp_unprotect
srtp: estimated u_packet index: 0000000000002537
srtp: computed auth tag:    442e409b1e673746a0d1
srtp: packet auth tag:      442e409b1e673746a0d1
sending word: hi
srtp: function srtp_unprotect
srtp: estimated u_packet index: 0000000000002538
srtp: computed auth tag:    dd955a5d81cceed63c04
srtp: packet auth tag:      dd955a5d81cceed63c04
sending word: hi
srtp: function srtp_unprotect
srtp: estimated u_packet index: 0000000000002539
srtp: computed auth tag:    85d9ea33c13d03d01848
srtp: packet auth tag:      85d9ea33c13d03d01848
sending word: hello
srtp: function srtp_unprotect
srtp: estimated u_packet index: 000000000000253a
srtp: computed auth tag:    f9197cc837bbfe81ad37
srtp: packet auth tag:      f9197cc837bbfe81ad37
sending word: hello
srtp: function srtp_unprotect
srtp: estimated u_packet index: 000000000000253b
srtp: computed auth tag:    6cf09e71c395cbdad2fc
srtp: packet auth tag:      6cf09e71c395cbdad2fc
sending word: hello
```

Figure 6. The Results of Client Side, Where Stream of Words sent from Server in been Received at Client Side in a Secured Manner


student@sascmp-175:~/shashi/srtp
File Edit View Terminal Tabs Help

```
[student@sascmp-175 srtp]$ set k='test/rand_gen -n 30'
[student@sascmp-175 srtp]$ echo $k cleec3717da76195bb878578790af71c4ee9f859e197a
414a78d5abc7451
cleec3717da76195bb878578790af71c4ee9f859e197a414a78d5abc7451
[student@sascmp-175 srtp]$ test/rtpw -s -k $k -ea 192.168.12.112 9999
usage: test/rtpw [-d <debug>]* [-k <key> [-a][-e]] [-s | -r] dest_ip dest_port
or     test/rtpw -l
where  -a use message authentication
       -e use encryption
       -k <key>  sets the srtp master key
       -s act as rtp sender
       -r act as rtp receiver
       -l list debug modules
       -d <debug> turn on debugging for module <debug>
[student@sascmp-175 srtp]$ test/rtpw -s 192.168.12.112 9000
security services: none
\tkhdts�^{ ^^ \ ^ _�^ c ^ f�^ j!^ m�^ q)^ t�^ x1^ {�^ ∞9^ ��^ �A^ ��^ �I^ ��^ �Q
^ ˆ^ � ^ ə^ � ^ ⅃^ �%^ ��^ �-^ ��^ �5^ ��^ �=^ ��^ �E^ ��^  M^  �^U^
                                                                  �^ ]^  �^
e^  �^  m^  �^ $u^ '�^ +}^ / ^ 2�^ 6    ^ 9�^ = ^ @�^ D ^ G�^ K!^ N�^ R)^ U�^ Y1
^ \�^ `9^ c�^ gA^ j�^ nI^ q�^ uQ^ x�^ |Y^ ∞�^ �a^ ��^ �i^ ��^ �q^ ��^ �y^ ��^ ��
1> 4B :H 9G >P 7HS$6K =T(@Q'>F JL$JQ'GN$EP%GT(CS$;M DX'J^-H\)FZ'HZ(BT";K =M =K B
P Rc/Uf2CV BU G\#>S Tg/K^&GX"Sa.S_-Yb3dk?NW*LZ'HY%M_-Qc1Qc1Tf4Rd2FX&K`(Sh/Uj/Oe%
Ib Ic C] <S
          Lc K` I` Kb Mf$Ng%Ke$Hb!Rl-D^ Ld(Rj0Yn5Yn6Rd0]n:ds<]j4Ve.Xg0Zi2Yh1Xi2
\m6Pc+Yl4fyASf.Mb)<Q AV =P O](_l7Ra(FU Rd&Wi+Od#Sh'_u4Mc"Sl+>V Rl-\u;Qm2Fb'B\ Ke
$7Q Jd#Id!Ql)Rk)G` Le$Qj)G] >S Rg,i~Ei~Fbt@cq@\h:R^.GS#P\*EQ Vc-Q^(:Gr∞G]k.dr5Tb
$Tb$bq0O] Yk-<N 8G FT!Sc2=O K_*E^%Jf+Qm0Ke(H[#OY%JK LF"]Y4DM FV%N[-T`2U`5NW,MT*Z
a7^g<`i>Tb1CQ <O DW DY <Q 9M G[*Pa5IY2BP-DR1NY9R];MY5DP*7F BP,8H$DS49I.2B'@O0<J)
8C#07 LR6DH->G,<E*DQ5AN2\iMFP5bfKYY?YS;\U
```

Figure 7. MPEG Video Clip Data Sending From Server to Client


student@sascmp-175:~/shashi/srtp
File Edit View Terminal Tabs Help

```
[student@sascmp-175 srtp]$ set k=cleec3717da76195bb878578790af71c4ee9f859e197a41
4a78d5abc7451
[student@sascmp-175 srtp]$ test/rtpw -r -k $k -ea 192.168.12.112 9999
usage: test/rtpw [-d <debug>]* [-k <key> [-a][-e]] [-s | -r] dest_ip dest_port
or     test/rtpw -l
where  -a use message authentication
       -e use encryption
       -k <key>  sets the srtp master key
       -s act as rtp sender
       -r act as rtp receiver
       -l list debug modules
       -d <debug> turn on debugging for module <debug>
[student@sascmp-175 srtp]$ test/rtpw -r 192.168.12.112 9999
security services: none
test/rtpw: socket bind error
Address already in use
[student@sascmp-175 srtp]$ test/rtpw -r 192.168.12.112 9000
security services: none
\tkhdts�^{ ^^ \ ^ _�^ c ^ f�^ j!^ m�^ q)^ t�^ x1^ {�^ ∞9^ ��^ �A^ ��^ �I^ ��^ �Q
^ ˆ^ � ^ ə^ � ^ ⅃^ �%^ ��^ �-^ ��^ �5^ ��^ �=^ ��^ �E^ ��^  M^  �^U^
                                                              �^ ]^  �^
e^  �^  m^  �^ $u^ '�^ +}^ /9�^ = ^ @�^ D ^ G�^ K!^ N�^ R)^ U�^ Y1^ \�^ `9^ c�^
gA^ j�^ nI^ q�^ uQ^ x�^ |Y^ ∞�^ �a^ ��^ �i^ ��^ �q^ ��^ �y^ ��^ mdat#2 ,; 2@ -;
1> 4B :H 9G >P 7HS$6K =T(@Q'>F JL$JQ'GN$EP%GT(CS$;M DX'J^-H\)FZ'HZ(BT";K =M =K B
P Rc/Uf2 Tg/K^&GX"Sa.S_-Yb3dk?NW*LZ'HY%M_-Qc1Qc1Tf4Rd2FX&K`(Sh/Uj/Oe%Ib Ic C] <S

Lc K` I` Kb Mf$Ng%Ke$Hb!Rl-D^ Ld(Rj0Yn5Yn6RVe.Xg0Zi2Yh1Xi2\m6Pc+Yl4fyASf.Mb)<Q A
V =P O](_l7Ra(FU Rd&Wi+Od#Sh'_u4Mc"Sl+>V Rl-\u;Qm2Fb'B\ Ke$7Q Jd#Id!Ql)Rk)G` Leg
,i~Ei~Fbt@cq@\h:R^.GS#P\*EQ Vc-Q^(:Gr∞G]k.dr5Tb$Tb$bq0O] Yk-<N 8G FT!Sc2=O K_*E^
%Jf+Qm0Ke(H[#OY%JK LF"]Y4DM FV%N[-*Za7^g<`i>Tb1CQ <O DW DY <Q 9M G[*Pa5IY2BP-DR1
NY9R];MY5DP*7F BP,8H$DS49I.2B'@O0<J)8C#07 LR6DH->G,<E*DQ5AN2\iMFP5bfKY�����������
 �����rB^J1fU9k\?g\@`T:j��x��w��|��y�����j����sabR6(9 %&!
```

Figure 8. MPEG Video Clip Data Receiving From Server by Client

Figure 9. Frame of Origional Video Clip

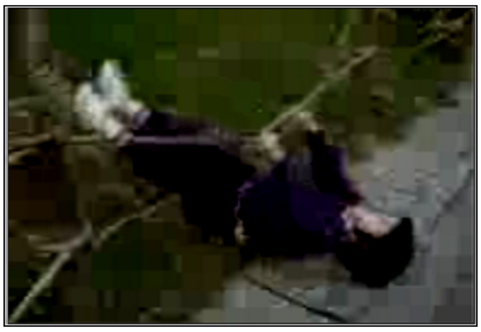


Figure 10. Frame of Encrypted Video Clip

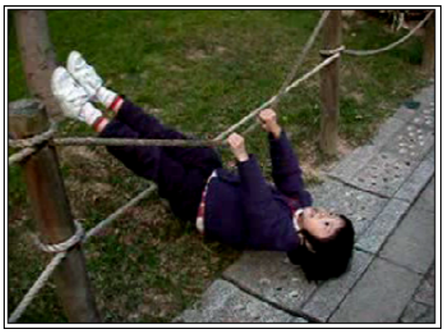


Figure 11. Frame of Video Clip after Decryption

7. TESTING

Table 1. Test Drive and its Functions

Test Driver	Function Tested
kernel_driver	crypto kernel (ciphers, auth funcs, rng)
srtp_driver	Srtp in-memory tests (does not use the network)
rdbx_driver	rdbx (extended replay database)
roc_driver	extended sequence number functions
replay_driver	replay database
cipher_driver	ciphers
auth_driver	Hash functions

Several test drivers and a simple and portable srtp application is included in the test/ subdirectory. Some of the test driver and its Function tested using this libsrtp are listed above in table 1.

The application for media client_server.c is a simple rtp application which reads data from particular file and then sends them out one at a time using [s]rtp. Manual srtp keying uses the -k option automated key management using gdoi will be added later.

The usage for client_server.c is

client_server [[-d <debug>]* [-k <key> [-a][-e]] [-s | -r] dest_ip dest_port][-l]

Either the -s (sender) or -r (receiver) option must be chosen. The values dest_ip, dest_port are the IP address and UDP port to which the dictionary will be sent, respectively. The options are:

- **s** (S)RTP sender - causes app to send data
- **r** (S)RTP receive - causes app to receive data
- **k** <key> use SRTP master key <key>, where the key is a hexadecimal value (without the leading "0x")
- **e** encrypt/decrypt (for data confidentiality) (requires use of -k option as well)
- **a** message authentication (requires use of -k option as well)
- **l** list the available debug modules

- d <debug> turn on debugging for module <debug>
Since this is the receiver side (client) the usage of client should be the usage for client_server.c is
client_server [[-d <debug>]* [-k <key> [-a][-e]] [-s] dest_ip dest_port][-l]

8. CONCLUSION

SRTP has been used for secure transmission of multimedia data from one PC to another PC. The multimedia has been secured using AES algorithm. The technique applied for encryption/decryption is lightweight. The encryption and the decryption of the multimedia GOP have been achieved successfully.

The sending and receiving of responses from the client and server are done in the UDP/IP-based transport layer. With any conceivable personal computer based security model, a dedicated hacker can, with sufficient effort, successfully attack a particular piece of content. This is unavoidable if the media is to be rendered on a system with an open architecture (such as a personal computer) where the attacker controls the system. Under the proposed secure streaming media system, the amount of work required for such an attack would be significant. However, the real strength of this approach is that the overall system will survive even when individual pieces of content are successfully hacked.

REFERENCES

[1] William Stallings, Cryptography and Network Security: Principles and Practices, Prentice Hall, 2003.
[2] Charlie Kaufman, Radia Perlman, and Mike Speciner, Network Security: PRIVATE Communications in a PUBLIC World, Prentice Hall, 2002.
[3] The Secure Real Time Transport Protocol, IETF draft, http://www.globecom.net/ietf/draft/draft-ietf-avt-srtp-00.html
[4] Secure RTP, http://www.vovida.org/protocols/downloads/srtp/
[5] D. Cohen. "Specifications for the Network Voice Protocol", Network Working Group, RFC 741, November 1977.
[6] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, R. Sparks, M. Handley, and E. Schooler. "SIP: Session Initiation Protocol", Internet Engineering Task Force, RFC 3261, June 2002.
[7] H. Schulzrinne, A. Rao, and R. Lanphier. "Real Time Streaming Protocol (RTSP)", Internet Engineering Task Force, RFC 2326, April 1998.
[8] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. “The Secure Real-time Transport Protocol (SRTP)”, IETF RFC 3711, March 2004.
[9] Coded Representation of Picture, Audio and Multimedia/Hypermedia Information. Committee Draft of standard ISO/IEC 13818-2, 2000.
[10] Confidential multimedia communication in IP networks: Communication Systems, Volume 1, Issue, 25-28 Nov. 2002 Page(s): 516 - 523 vol.1
[11] Jayshri Nehete, K. Bhagyalakshmi, M. B. Manjunath, Shashikant Chaudhari, T. R. Ramamohan, “A Real-time MPEG Video Encryption Algorithm using AES”, for NCC-2003.
[12] Andreas Uhl and Andreas Pommer, “Image and Video encryption From Digital Rights Management to Secured Personal Communication”, Springer, 2005.
[13] George Anastasios Spanost and Tracy Bradley Maples, “*Performance Study of a Selective Encryption Scheme for the Security of Networked, Real-Time Video*”, Proceedings of the Fourth International Conference on Computer Communications and Networking, Las Vegas, Nevada. October 1995.
[14] William Stalling, “Cryptography and Network Security Principles and Practices”, Pearson education, 2003.
[15] Network Working Group, M. Baugher, Request for Comments: 3711, D. McGrew, March 2004.
[16] Siddique S, Ege R.K, Sadjadi. S.M., “*Multimedia Communication*”, Proceedings of IEEE Transactions on Modeling of multimedia streaming, pp.271-276, April 2005.
[17] Dapeng Wu Hou Y.T, Wenwu Zhu, Ya-Qin Zhang, Peha J.M, “Streaming Video Over Internet: Approaches and Direction”, *IEEE Transactions On Circuits and systems for Video Technology*, Vol. 11, Issue no. 3, pp. 282-300, March 2001.
[18] Fortino G, Russo W, Zimeo E, “Enhancing cooperative playback systems with efficient encrypted multimedia streaming”, *IEEE Transactions On multimedia and expo*, Vol. 2, Issue no. 6, pp. 57-60, July 2003.
[19] Lo Iacono L., Ruland C, “Confidential multimedia communication in IP networks”, *IEEE Transactions On Communication Systems*, Vol. 1, Issue no. 25, pp. 516-523, November 2002.
[20] Hess A, Klaue J., “A video-spam detection approach for unprotected multimedia flows based on active networks”, *IEEE transactions on multimedia security*, Vol. 30, Issue no. 31, pp. 461-465, Sep 2004.
[21] Xinmiao Zhang, Parhi K.K, “Implementation approaches for the Advanced Encryption Standard algorithm”, *IEEE Transactions On Circuits and Systems Magazine*, Vol. 2, Issue no. 4, pp. 24 – 46, Oct 2002.
[22] Kaindl, M. Hagenauer, J. “*Multimedia data transmission over mobile Internet*” Volume: 4, on pp 2015- 2019, Feb 2006.
[23] Kami H., Kurashige T., Higashi M., Imaide T., Kinugasa T. “Consumer Electronics”, *ICCE. International Conference on*, pp 82 – 83, 1999.