STREAMING MEDIA SECURITY USING DIGITAL RIGHTS MANAGEMENT

A Thesis

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

by

Deepali Holankar

December 2003

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Mark Stamp

Dr. Chris Pollett

Dr. Xiao Su

APPROVED FOR THE UNIVERSITY

ABSTRACT

STREAMING MEDIA SECURITY USING DIGITAL RIGHTS MANAGEMENT

By Deepali Holankar

Standards for streaming media technology are available in the areas of security and privacy but integrity and replay protection are still areas of ongoing research. This thesis aims at studying the standards and providing a viable solution for security in streaming media technology with implementation.

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.

We apply the concepts of digital rights management to streaming media in order to solve integrity and replay problems within reasonable limitations.

ACKNOWLEDGEMENTS

I would like to thank Professor Mark Stamp for his guidance, patience and insights without which my thesis would not have been possible.

INDEX OF CONTENTS

APPENDICES

INDEX OF TABLES AND FIGURES

1 Introduction

Digital rights management for streaming media deals with policy enforcement issues. An introduction and overview of digital rights management is given followed by problems in streaming media securely and an overview of a streaming media system. The streaming media is compared to digital pay TV system and similarities and differences in approach to securing streaming media versus digital pay TV are considered.

Background information for streaming multimedia and typical audio/video conferencing scenarios are considered. Followed by the design issues of a secure streaming media system, implementation and deployment cases.

1.1 Digital rights management

Digital rights management (DRM) attempts to provide for the secure delivery of digital content with restrictions on the usage of the content after delivery. For example, the provider of a piece of digital content might want to restrict the end-user’s ability to duplicate the information. Such restrictions are necessary if the provider is to maintain any control on the distribution of the content. In contrast to classic cryptography, which aims to protect against an unintended recipient, the protection provided by a DRM system is primarily aimed at the legitimate recipient. When seen in this light, it is clear that cryptography is only a very small part of a DRM solution.

In addition, DRM protection must stay with the content even after delivery. In the DRM literature, this required level of protection, which goes beyond the protection that standard cryptography can provide, is often referred to as "persistent protection". Background information on DRM, including an outline of a complete DRM system is in Stamp M., (2003).

Consider a scenario in which company A wants to stream a live baseball game to N clients. Company A does not want its competitor, company B, to hack their media stream and add noise or distortion to the signal. Moreover Company A does not want any of its clients to record the game and redistribute it. Company A only wants to allow paying customers to have access to the media stream. Digital rights management is designed to deals with such issues.

The proposed model for secure streaming media described in this thesis employs some features commonly used by digital rights management systems. In the streaming media scenario, these features primarily provide replay protection, which is lacking in current approaches to streaming media delivery. Of course, any media streamed to a personal computer can be recorded using an analog device. For example, the video displayed on the monitor screen can be captured via screen shots. In the DRM world, this fundamental problem is known as the "analog hole" as referred by Doctorow C., (2003) and is considered beyond the boundary of protection provided by a DRM system. The DRM philosophy is to make an attack on the system as difficult as possible, while realizing that perfect protection cannot be achieved in the current computing environment.

The proposed secure model also does not deal with the storing of streamed data on hard disk or other such media. The security issues concerning storing of copyrighted material and its reproducibility are separate issues that are not the concern of this thesis.

In addition, this thesis does not contain a detailed discussion of key management issues. Key management techniques are well established; see, for example, Kaufman C., (2002) for further information.

What this thesis does provide is a proposed technique to achieve a measure of replay protection and message integrity for streamed media. Of course, it is possible to garble or destroy the encrypted media stream, thus rendering it useless to the legitimate recipient. This thesis does not discuss specific protection against such malicious attacks.

1.2 Problem definition

The problem can be simply defined as follows: when the media stream is in transition between two endpoints, it should not be possible for a third party to play the same media stream by simply capturing it. Once the data stream has reached its desired endpoint, the audio and video device driver at the endpoint should be able to play the media stream, so that the media stream is audible and visible to the participating people.

If the endpoint decides to record the media stream, they should be able to do so provided that they have made payments for its copyright license, but should not be able to change the media stream in any manner. Any attempt to change the media stream should make it useless for everyone, thus making it obvious that the media data has been tampered with. Also, if an attacker was able to hack into one session that should not imply that he is now able to hack into any sessions held by the same party.

1.3 Possible attacks on a secure streaming media system

In this section, different scenarios of possible attack on a secure streaming media system are discussed.

Scenario 1:

During transition between two endpoints, an attacker might spoof the stream of data passing by. If the attacker obtains the encrypted stream of data, he should not be able to decrypt the stream and get the raw data.

Scenario 2:

A man in the middle attack should not be possible. An attacker in the middle should not be able to be pretending to be an endpoint.

Scenario 3:

Once the media stream reaches the endpoint, it should be impossible for any person at the endpoint to decrypt the stream media, except for the trusted media player software.

Scenario 4:

Flaws in operating system security should not affect the security of the streaming media system.

Scenario 5:

Compromise of a single piece of software at one endpoint, should not allow the attacker to hack into the entire system and compromise all parties involved.

Scenario 6:

Unauthorized software should not be able to steal the data stream between the point of decryption and the point where the decrypted data is sent to video or audio codec.

Scenario 7:

Buffering mechanisms in the system should not allow decrypted data to be copied intermittently, which could then be played by unauthorized systems.

Scenario 8:

Once the end-user has played the media stream, he should not be able to record it on any unauthorized media.

Scenario 9:

Once the end-user has played the media, he should be allowed to play the media stream again with valid authorization.

Scenario 10:

Special care has to be taken to enable video control options such as replay, forward, stop and pause. These options would allow the user to go back in the encrypted stream.

Scenario 11:

Packet loss between two endpoints should not invalidate the entire media stream.

1.4 Comparison with digital pay TV

Digital Pay TV has a subscriber management technique where it maintains a list of subscribers and their access rights. A subscriber may choose to upgrade or downgrade his viewable list of channels by paying more or less money every month or upgrading. It is also possible that a subscriber will not pay his monthly fee. In such a case, the digital pay TV system will downgrade the subscriber's viewable list. So, for a given subscriber the list of viewable channels may not remain constant across different sessions, but each session does not necessarily dictate a change in the privilege rights.

In multimedia conferencing, the capabilities of each entity are negotiated at the beginning of each session. A principle entity that owns the session can decide whether a particular entity should be allowed to participate in the session or not.

The Digital pay TV system assumes that a valid subscriber will use a particular legitimate set-top box from the provider. In multimedia conferencing, there is no such predefined set-top box. Instead security is based on the assumption that the audio and video device drivers used by different people support the security features implemented for replay and integrity protection.

Digital pay TV employs proprietary algorithms for scrambling data, but the same is not true for multimedia conferencing. The standards for multimedia conferencing are open, and they are available to anyone, who may want to implement a compatible system.

The following Table 1 compares Digital Pay TV and streaming media systems.

Comparison Features	Digital Pay TV	Multimedia Conferencing
Management of Privileges	Fixed and verified from subscription list	Changes with each session. Each session has flexibility and different privileges.
Available Box	Legitimate set-top box needs to distinguish between valid and hackers with illegal boxes	Uses audio and video codecs with security and encryption features.
Algorithms	May be proprietary	Would have to be compatible with different multimedia streaming standards
Transferring Media	Method is usually broadcast, accessible to everyone	Sent by using network layer protocols, reaches individual IP addresses. (Packets can be sniffed between two endpoints)
Authentication	Validity of the set-top box & receiver	Authenticate parties at initiation.

Comparison Features	Digital Pay TV	Multimedia Conferencing
Facility to change software	Software of all the set-top boxes can be upgraded once an attack or intrusion has been detected.	In each session the desired security level must be determined and appropriate encryption algorithm chosen.
Packets Lost in transmission	Encryption and decryption done assuming no packet loss between transmitter and receiver	Packet loss between two endpoints must be considered and the security mechanism should not fail if such a loss occurs.
Speed Factor	Encryption and decryption does not significantly affect the performance since implemented in hardware	Encryption and decryption may affect performance, as actual bandwidth and throughput varies and depends on network congestion.

- **Table 1 Comparison of digital pay TV and multimedia conferencing**

Digital pay TV does not care about the identity of the person watching the setup box as long as the setup box is legitimate whereas in multimedia conferencing the identity of each entity has to be well established before the session can begin. One person should not be able to impersonate as another person.

Anderson R., (2001) gives additional details of digital pay TV.

2 Background

This background section gives detailed information on streaming multimedia and security concerns. The information given helps in understanding the proposed security.

2.1 Streaming multimedia components

The following topics are discussed in this section.

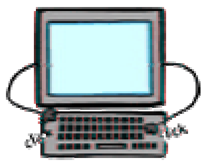
- Audio/video conferencing
- Streaming stored media
- Streaming live or interactive media

2.2 Audio or video conferencing

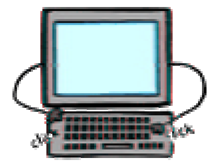
The basic goal of audio or video conferencing is transmission of real-time audio, video and data communications over packet-based networks in real-time. Following are the basic components, protocols and procedures for providing multimedia communication over packet-based networks.

The four basic components are as follows:

1. Terminals
2. Gateways
3. Gatekeepers
4. Multipoint control units (MCUs).



← Packet based network →



•**Figure 1 Terminals**

H.323 network→ [Gateway] ←Non H.323 network

For example,

PC Net meeting→ [Gateway] ←Public switched telephones

•**Figure 2 Gateway**

H.323 terminal 1 → [Gatekeeper] ←H.323 terminal 2

Within H.323 network, gatekeeper has the following functionality: Addressing, authorization and authentication of terminals and gateways, Bandwidth management, accounting, billing and charging

•**Figure 3 Gatekeeper**

H.323 Terminal 1→ [Multipoint Control Units] ← H.323 Terminal 2

•**Figure 4 Multipoint control units**

The basic terminology given below is taken from H.323 Definitions (2003):

H.323 Zone is a collection of all terminals, gateways and multi-control units managed by a single gatekeeper.

Audio Codec encodes the audio signal from the microphone for transmission on the transmitting terminal and decodes the received audio code that is then sent to the speaker on the receiving terminal. Minimum service provided by H.323 is audio and so all terminals must have at least one audio codec support.

Video Codec encodes video from camera for transmission on the transmitting terminal and decodes the received video code that is then sent to the video display on the receiving terminal.

Registration, Admission, and Status (RAS) is used to perform registration, admission control, bandwidth changes, and status and to disengage procedures between endpoints and gatekeepers. An RAS channel is used to exchange RAS messages. This signaling channel is opened between an endpoint and a gatekeeper prior to the establishment of any other channels.

Call Signaling is used to establish a connection between two endpoints. This is achieved by exchanging protocol messages on the call-signaling channel.

Control Signaling is used to exchange end-to-end control messages governing the operation of the endpoint. Control messages carry information related to capabilities exchange, opening and closing of logical channels used to carry media streams, flow-control messages, and general commands and indications.

Real-Time Transport Protocol (RTP) provides end-to-end delivery services of real-time audio and video. RTP provides payload-type identification, sequence numbering, time stamping, and delivery monitoring. On IP-based networks RTP is used together with UDP.

Real-Time Transport Control Protocol (RTCP) primarily provides feedback on the quality of data distribution. Functions include carrying a transport-level identifier for an RTP source, called a canonical name, which is used by receivers to synchronize audio and video.

2.3 Available open source resources

Following are the available open source resources:
The website http://www.openh323.org has the entire H.323 protocol stack available as open-source. With each passing day additional audio and video codecs are included in the project. For secure transmission of media stream between two endpoints the IETF has proposed a standard for secure RTP that

is still in draft format at http://www.ietf.org/. An implementation of secure RTP in the open source world is available from Cisco at the sourceforge website. http://srtp.sourceforge.net.

Secure transmission of a media stream can also be done using IPSec by changing the security policy at each endpoint. An implementation of IPSec is available at www.freeswan.org. Media encryption techniques are available and well established by using sRTP or IPSec Layer.

Secure key exchange and management can be done using Deffie-Hellman key exchange and RSA (which is available in patent-free form). As media encryption techniques are well established, it can be safely assumed that media stream is sent from one endpoint to another in secure fashion.

2.4 Digital rights management for streaming media

A typical digital rights management system for streaming media may be broken into the following basic components:

- Key Management / Licensing Issues
- Encryption of streaming media
- Encryption of control information
- Policy Enforcement

The port for secure communications should be different from the port used by insecure communications (analogous to insecure http on port 80 and HTTPS on 443). Control parameters may or may not be sent in an encrypted channel. This decision is negotiated in the initial handshake. Exchange of certificates occurs to establish the identity of each entity. The negotiated handshake and exchange of certificate occurs before any other exchange of messages.

A session may run in three different modes:

- authentication only
- encryption only
- authentication and encryption

Multipoint procedures will negotiate independently with each channel the encryption algorithm. So for a given session it is possible that different encrypted streams are going to different channels.

Authentication may be done using Diffie-Hellman with optional authentication or subscription-based authentication (for e.g. symmetric encryption, hashing, certificate-based signatures).

It would be good to have media encryption procedures accelerated in order to have minimal impact on the quality of service due to the features.

There is a need to define a principal owner or entity for each multimedia session or conference. This entity may keep a list of anticipated participants and the security level for each participant.

The functionality of a smart card could be achieved by using a tamper-resistant hardware or software module, which would generate keys for each session.

There is also a need for an access-control guard module at each entity or end point identifying the session and the entity uniquely. This module would have the responsibility to ensure that the media stream data is not recorded by insecure means. This would also monitor all the network identification cards on the system, to monitor any suspicious activity.

It is essential to generate unique session keys for each session. To generate a rollover counter that exceeds the length of 64 bytes, each entity will exchange rollover counters before generating their session keys. This will prevent the repetition of session keys until after 2^512 sessions.

2.5 Key management and licensing issues

Following is the proposed flow between two endpoints:

- Initial Messages

- Request Privacy System

The sender of this message wishes to use an encryption system. It will wait for the receiver of this message to send the same message back.

- Cannot Encrypt

Sent in reply to the above message, saying the sender will not use an encryption system.

- Failure to start an encryption system

The sender has failed to start its encryption system, which may be due to key exchange failure. For security reasons, this is sent without giving the cause of the failure.

2.5.1 Session key exchange

The session key consists of the following:

- 8-bit message identifier
- Initialization vector with error correction
- 4N-bit random value where the value of N depends on the encryption algorithm used

Sender has four random values of N-bits T1,T2,T3 and T4. Receiver also has four random values of N-bits R1,R2,R3 and R4.

Sender Key 1: T1 xor R3

Sender Key 2: T2 xor R4

Receiver Key 1: T3 xor R1

Receiver Key 2: T4 xor R2

Key1 is used for encryption of the control signals and frame control and key2 can be used alternatively for encrypting media stream.

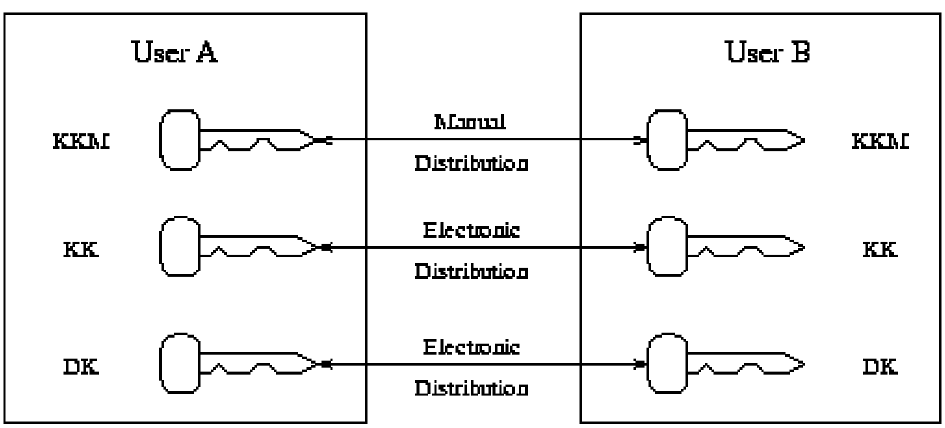
2.6 Key distribution environments

Following are different types of key distribution environments:

- Point to point
- Key distribution center
- Key translation center

2.6.1 Point to point

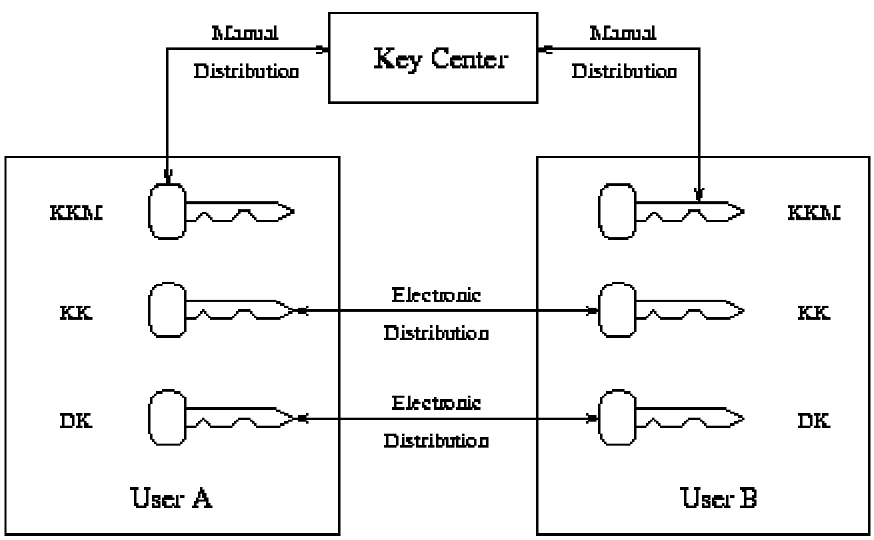
Point to point is a two-layer environment, where the two terminals share a common key. Point to point is the simplest of all key environments, moreover point to point does not involve any additional hardware or software to manage keys.



•**Figure 5 Point to point**

2.6.2 Key distribution center

The definition key distribution center as given in Key definitions (2003) is "The Key Distribution Center generates keys for its users. If an originator wants to send an encrypted message to a recipient, the originator submits the request to the Key Distribution Center. The Center generates and returns two identical keys to the originator. The first key is encrypted using the KKM shared between the Center and the originator. The originator decrypts the key, and uses it to encrypt the message. The second key is encrypted using the KKM shared between the Center and the recipient. The originator transfers this key electronically to the recipient. The recipient decrypts the key, and uses it to decrypt the originator's message."



• **Figure 6 Managed key distribution**

2.6.3 Key translation center

The definition of a key translation center (KTC) as given in Key definitions (2003) is "Key Translation Centers are used when two parties require the key management functions provided by the center, but one or both of the parties want to generate the KKs and DKs. In this scenario, the originator submits a key and the recipient name to the Center. The Center encrypts the key using the KKM shared between the Center and the recipient, and returns the encrypted key to the originator. The originator transfers the key electronically to the recipient."

Messages for authentication may be defined as follows:

- Authentication initiation
- Authentication response
- Authentication complete
- Authentication failed

Authentication can be done using the following well-known methods:

- Diffie-Hellman key exchange
- RSA based operation
- Authentication using certificates and digital signatures

Following sections 2.7 to 2.8 are inspired by H.235 Security Protocol suite (2003).

2.7 Encryption of streaming media

The encryption of streaming media is mainly implemented using secure RTP. The following are its main features:

- Preserves cRTP efficacy
- Minimal packet expansion
- Low computational cost
- RTP and RTCP security is provided by secure RTP.
- Basic Operation includes
- Confidentiality of RTP data
- Authentication of RTP header and data
- Protects against replay and denial of service (DoS) attacks
- All the protections are applied to control channel RTCP as well.

The following details as discussed in the white paper describing secure RTP from http://srtp.sourceforge.net (2003).

	SRTP	ESP
Authentication	Only for the RTP headers	At RTP, UDP and ESP levels
Header	Smaller than ESP	Larger than SRTP
Performance issues	Less encrypted data is sent, accelerated SRTP is also available.	More performance issues as more encrypted data is sent, slowing the system

•**Table 2 Comparison of SRTP and ESP**

- Methods used by SRTP

- Encryption uses AES-128 in Counter Mode
- Authentication uses TMMHv2 (authentication tag is encrypted value of a universal hash)

2.8 Encryption of control information

Following are the typical control messages sent between two endpoints.

- Open logical channel
- Open logical channel acknowledgement

Control messages are sent after the initial setup messages and session keys are established.
Control messages can negotiate the encryption methods for the media stream channel.

2.9 Decoding at receiver side

Decoding at the receiver side is vulnerable to a possible attack, where the decoded stream may be captured. To avoid such an attack a reasonable solution is to avoid decoding until the very last moment. It should not be possible for an attacker to get the decoded stream easily, after all the effort spent in encrypting the media stream and transferring it over the network.

There are security issues related to buffering of the decoded data before giving it to the media codec.

Parameters exchanged between the sender and receiver can utilize individuality built into each video codec. This could be equated to Lamport's hash algorithm, which generates a unique password using salt and sequence number.

Sender scrambles the data according to the receiver parameters. The sender should have the capability of sending media stream in different scrambled streams depending on the de-scrambling algorithms supported by different receivers. The codec should take as input the scrambled stream and play the de-scrambled stream directly to the end-user.
It should also be possible to use selective encryption on some important piece of information. The start of encryption and end of encryption need to be marked in the media stream. Each packet has initialization vector needed to decrypt the authentication header in each unit. This would avoid rendering the data useless on a single packet loss.

3 Design

Digital rights management (DRM) is an evolving field. The current DRM market includes many proprietary systems as well as open source solutions such as Media-S (2003). However, the majority of DRM systems focus on licensing management and rights. Though license management constituents are an important part of a DRM, it cannot yield a secure system by itself. DRM can only be successful if there exists a complete security chain that begins with the data transmission and persists beyond the point where the client accesses the content. We have incorporated certain DRM features into our proposed secure streaming media system. These features include license management, and several additional security features as discussed below.

This thesis is focused on enhancing the rights enforcement ability for streaming media on the client. Of course, it is crucial that the entire system be secure in order to avoid a weak link in the process. Therefore, an overview of the entire system is essential, but detailed emphasis is placed on the client software, since that is the primary contribution of this thesis.

3.1 Generic model

First, a generic model for a streaming media system is described. The basic components of such a model would include the following

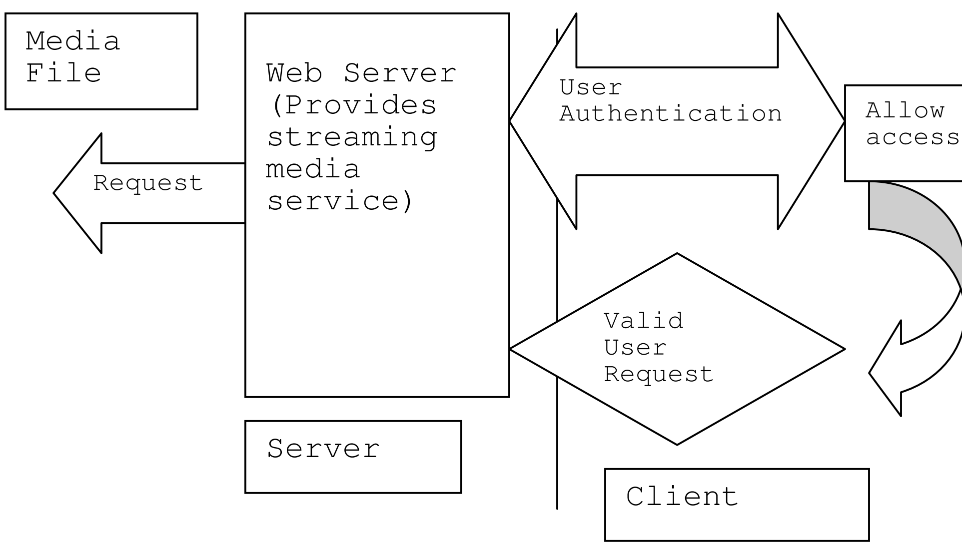
- Streaming web server
- Authentication protocol on web server and client
- Client web browser to request media
- Client application to receive media data (e.g. Real Player plug-in)
- Client library interface between kernel (device driver) and user space (application)

- Client device driver to utilize media data

The basic components of this generic model are shown in a simplified block diagram form in Figure 7.

This generic system would function in the following manner:

- The web server offers streaming media services
- A client requests a media file from the web server
- The web server authenticates the user and the user authenticates the web server (mutual authentication)
- On successful mutual authentication, the web server employs RTP to stream the data from the server to the client
- The client web browser opens the default media application (e.g. Real Player or windows media player)
- The media application strips the RTP header and sequences the packets



•**Figure 7 Generic model**

- The media application uses system calls or library functions to write the data to the device that plays the file

- The device driver stores the data in its internal memory, using interrupts (or other procedures) to write the data to the appropriate port.

The above system has several security vulnerabilities. For example, the streamed data can be captured at any point between the two endpoints and the resulting data is subject to replay. Moreover, there is no protection to prevent the client side from capturing and redistributing the data to others.

3.2 Proposed secure model

The proposed security model includes the same basic components as the generic streaming media system discussed in the previous section, with a few additional security features. For example, the web server includes a license manager to manage access to requested data. The operation of this feature will be described in more detail below.

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 (or uniqueness) 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. Therefore, homemade cryptographic algorithms that provide even minimal cryptographic strength will serve well as scrambling algorithms. For example, the tiny encryption algorithm (TEA), Wheeler D., Needham R., (2003) can be modified in many different ways to yield a large class of scrambling algorithms. While none of these modifications could be claimed to provide significant cryptographic strength, each could serve well as scrambling algorithms. The rationale behind scrambling is further discussed within the context of DRM in Stamp M., (2003).

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.

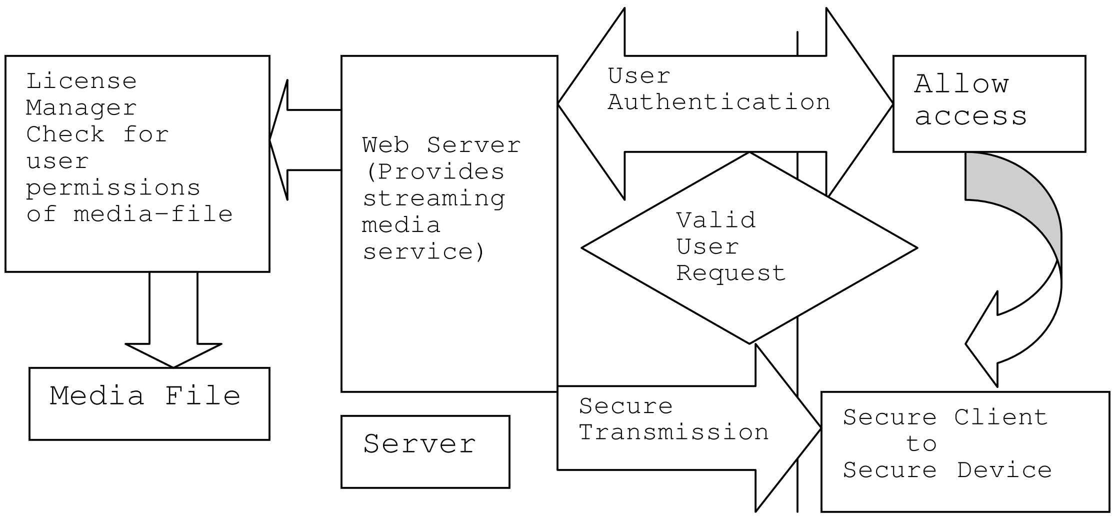
On the server side, the data is scrambled, and then encrypted. On the client side, the data is decrypted and the resulting scrambled data is passed to the media application. The media application passes the scrambled data to the secure device driver (discussed in more detail below), which de-scrambles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

Given these 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 secure web server
- The secure web server authenticates the user and the user authenticates the web server
- Upon successful mutual authentication, the web server gives the IP address of the client machine and client's username to its License manager. The client sends its encrypted list of supported scrambling algorithms to 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, see Schneier B., (1996)).
- 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 web browser opens the secure media application for the file.
- The media application requests the secure RTP decryption key. The user must authenticate in order for the client to obtain the decryption key. For example, authentication could be smart card based. Of course, any other user authentication method could be applied on the client.



•**Figure 8 Secure streaming model**

- The media application strips the secure RTP header and sequences the packets.
- The media application initializes its secure device driver with the ID number that specifies the scrambling algorithm used by the server, and the algorithm is initialized with the scrambling key.
- The media application is oblivious to the scrambling. It therefore writes the scrambled data to the device that plays the file.

- The secure device driver de-scrambles the data and writes plaintext data to the appropriate device buffer and port.

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
Transmit file by breaking into packets which are scrambled and encrypted	→	Receive file packets, Decrypt the packets Send to device driver for de-scrambling. Play the music.

- **Table 3 Flow of secure model**

The secure streaming media system is summarized in Figure 8.

The following table gives a simplified view of the interaction between the client and server in the secure model.

3.3 Comparison with a typical DRM system

Here follows a brief comparison of the security features in our proposed secure streaming media system with the features available in Windows Media Player (2003), a typical DRM system. The following six points are listed on Microsoft's website Windows Media Player (2003), as the primary security features of Windows Media Player. The implementation details in Windows Media Player are proprietary and not available to us. Our model differs from that of Windows Media Player in that it is more secure and robust as it has two layers of obfuscation making reverse engineering difficult. Descriptions of how our proposed system implements each of these features is given.

- Persistent Protection: The proposed model gives an individual license key to a client on a per transaction basis for each requested file. The protection is not only over the insecure network between client and server, but, due to the scrambling and the secure device driver, it persists all the way to the clients media device
- Strong Encryption: The proposed model uses secure RTP with 128-bit AES encryption. The model also employs scrambling, but it does not rely on scrambling for cryptographic security.

- Individualization: The scrambling algorithm is selected at random and the actual set of available scrambling algorithms is individual to each client. Therefore, the compromise of one client does not break the entire system. Moreover, the broken client can easily be replaced with an upgraded device driver that employs a different set of scrambling algorithms. The secure device driver could be made unique in other ways as well. For example, the methods discussed in Mishra P., (2003) or methods similar to those employed by metamorphic virus writers Balepin I., (2003) could be implemented. Such protections would certainly make the reverse engineering problem even more challenging for an attacker. This higher level of uniqueness needed for the same, has not been implemented, but it would clearly be feasible to do so.
- Secure Media Path: Proposed secure model does not de-scramble the media data until the last possible point in the process. The data passes through the entire system in scrambled form. Of course, over the insecure channel is further protected by strong encryption.
- Revocation and Renewal ability: Maintaining a revocation list with the License Manager would revoke compromised players. Revoked clients will then fail to authenticate with the License Manager. Moreover, if a particular scrambling algorithms is compromised, the server could simply avoid using the compromised algorithm.

Alternatively, all clients using the compromised algorithm could be upgraded with new device drivers that do not include that particular algorithm.

- Secure End-to-End Streaming and Downloads: Secure RTP is used for end-to-end streaming and downloads. The AES encryption algorithm (or other strong encryption algorithm) is used in secure RTP. Secure end-to-end transmission can also be accomplished using other well-established methods such as IPSec Kaufman C., (2002).

As can be seen from the proposed model and the discussion above, the License Manager is a significant part of the secure streaming media system. But it should be clear that license management is not the heart and soul of the system. The secure device driver and the uniqueness achieved via scrambling are the crucial security aspects of the system.

4 Implementation

The implementation section describes in detail the various components of the secure streaming media system. For a complete source code listing kindly see the appendices B to F.

4.1 Issues with different operating systems

The operating systems usually used in embedded multimedia devices are Real Time-Linux, Linux and VxWorks(article at http://www.linuxdevices.com/articles/AT3792919168.html by

Victor Yodaiken ELEC talk, June 2000 compares the three systems). The following passage attempts to compare implementation issues with the proposed secure streaming media model on these operating systems.

VxWorks does not have any memory protection between application and system tasks. This makes the device driver memory buffer in the proposed model available to any module through simple function calls. Moreover decryption of memory in the device driver adds overhead, making the process not so lightweight. If the decrypted media data is made available in the memory buffer only for periodic time slices, it makes it difficult for other hacker processes to contend for the decrypted data in the same time slice. The hacker process would need to synchronize with the frequency of available decrypted data. Moreover this process becomes difficult in a single processor system. It would be possible to break into the system in SMP system, but the task becomes many times difficult.

RT-Linux requires the user to divide the application into two distinct parts: the real-time part and the non-real-time part. The real-time part will be serviced rapidly, allowing it to meet deadlines, while the non-real-time part has full range of Linux resources available for use, but cannot have any real-time requirements. The division of multimedia data into real-time and non-real-time part should be carefully done. The constraints increase if the multimedia streamed is live and interactive. Decryption of media data on the device driver level increases the timing constraints of live and interactive data.

The implementation issues with Linux operating system are simplified, as Linux provides memory protection between kernel and user space. It also has non-swappable memory area for key material protection.

4.2 Open source resources utilized

The following open source resources were utilized in the implementation of the proposed security model

- http://www.openssl.org
- http:/www.acme.com/software/thttpd/thttpd_man.html
- http://srtp.sourceforge.net
- http://www.drfruitcake.com/linux/stest.html
- Linux device drivers tutorials
- Open source Intel audio driver i810_audio
- Open source crystal audio driver cs46xx

4.3 Distinct scrambling algorithms

Unique variations of the Tiny Encryption Algorithm Wheeler D. (2003) were implemented to generate a wide range of different scrambling algorithms. The current implementation supports about sixteen scrambling algorithms on the server side. The scrambling and de-scrambling code is compiled as separate object code, which can be easily linked with different sender and receiver programs. In practice, it would be easy to generate multiple receivers supporting different scrambling algorithms. For demonstration purposes, we have created three receivers wherein one is totally secure, one receiver is partially broken and another one is totally broken. These receivers demonstrate the functionality of the server negotiation and response when scrambling algorithms are hacked.

4.4 Minimal Hardware Requirements

To successfully implement and test the security model proposed in this thesis, it is essential to have the following

- Two personal computers running Linux kernel 2.4.16 or higher with sound cards and network adapters.
- Multimedia support should be enabled on the Linux operating system and sound driver configuration should be in modular mode.
- The sound card device driver should be available in open source.
- Both the personal computers should be connected with a network or crossover cable.

It should be noted that the performance results of the security model varies with the available processing power as well as the network card performance at both ends.

4.5 Implementation in Linux

This section describes the implementation details in Linux using kernel 2.4.16

4.5.1 Server components

HTTPS web server (tiny httpd server from http://www.stllinux.org/meeting_notes/2001/0719/tHTTPd/www.acme.com/software/thttpd/thttpd_man.html) was installed on the server. Open source open SSL at http://www.openssl.org was installed on the server side to support HTTPS. The makefile to compile HTTPS and openssl can be found in the appendix c. The server invokes cgi-scripts to start secure RTP streams. The default web page for the web server is also given in the appendix b. The function of the cgi-script is to get the environment settings of http username, http client IP address and requested file to invoke the sender program.

The sender program takes the following parameters

- Destination IP address
- Destination port
- Filename
- Username
- Sampling rate

The message sent by the server to the receiver has the following parameters in order:

- Session key
- Server IP address
- Server port
- Scrambling key
- Sampling rate of audio file
- Total number of packets

4.5.2 License Manager

The receiver sends a string of supported algorithms in encrypted form. The server randomly selects 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 is 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.

4.5.3 Receiver side components

The receiver side listens to a predefined secure RTP port. The server program sends a session key encrypted using the receiver's shared symmetric key. When the receiver gets the session key it sends the encrypted list of supported scrambling algorithms to the sender. The sender program chooses one of the scrambling algorithms and sends the multimedia data packets. The receiver application works in close communication with the receiver device driver. The receiver initializes the secure device driver using the selected scrambling algorithm and the scrambling key sent by the server. When the receiver is compiled, a private key for the receiver is built into the receiver executable file.

The receiver has an encrypted list of its supported scrambling algorithms which the sender decrypts to determine the scrambling algorithms supported by the device driver at the receiver end. If all the supported scrambling algorithms of the device driver are in the broken list maintained by the server, the device driver at the receiver end is considered hacked or broken. When the server receives a request of a hacked receiver, it immediately terminates the connection without explanation.

4.5.4 Secure Device Driver

The receiver application talks directly to the secure device driver. The secure device driver in turn talks directly to the secure device. The Linux artsd daemon used to monitor access to sound is killed to allow direct access to the sound device. The secure device driver is compiled using the de-scrambling algorithms and modifying the write function in the driver to de-scramble data before writing to the Direct Memory Access (DMA) buffer. The device reads the de-scrambled original data from the DMA buffer directly.

All Linux device drivers follow a uniform structure invoking read, write and setting the parameters (also known as ioctl calls). This makes the implementation of a secure device driver on different hardware platforms relatively simple under Linux. The secure driver implements an ioctl call to initialize the de-scrambling algorithm chosen with the de-scrambling key.

If any user tries to implement his or her own insecure device driver, the device driver will fail to understand the security parameters initialized by the application. The receiving application will immediately terminate, since the device driver does not understand security parameter. This case is a clear indication that something is wrong with the system.

4.5.5 Streaming data

The implementation on the receiver end uses two threads in round-robin mode.

- The receiving thread listens on a secure port for packets from the sender.
- The device driver thread begins writing to the sound device driver after receiving an initial buffer of data.

The implementation of streaming uses simple methodology. Sophisticated streaming with more control on startup latency and throughput can also be implemented with the proposed security techniques.

4.5.6 HTTPS and RTP clients

The HTTPS protocol is used in the secure model for username and password authentication, as well as client-server mutual authentication. The client request for a particular file is transmitted to the server using HTTPS. After user authentication, the server starts a secure RTP session for handshaking and streaming data transmission.

In practice, multimedia transmission is usually done using Real Time Transport Protocol (RTP) using UDP at the transport layer. The proposed secure model uses secure RTP for transmission between two endpoints.

A comparison between the three protocols secure RTP, RTP and HTTPS with respect to startup latency and throughput on available bandwidth is of interest. This comparison helps in understanding the performance penalty of the proposed secure streaming media model.

Simple HTTPS and RTP clients were implemented for the sole purpose of obtaining timing information.

4.5.7 Create receiver component

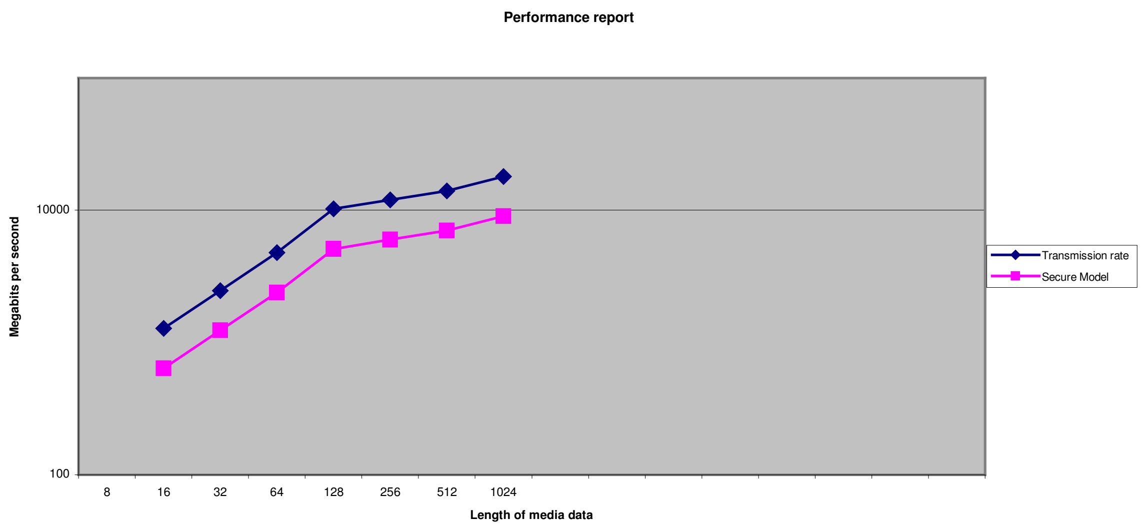
The create client program is used to automate the process of generating a shared symmetric key for the receiver and encrypted list of supported scrambling algorithms. The openSSL library function crypt, which implements DES illustrated Grabbe J., (2003) or MD5 Rivest R., (2003) encryption is used to encrypt the supported algorithms string in the receiver.

5 Deployment

This section describes in detail the test cases considered, performance data and analysis of the proposed system.

5.1 Performance penalty issues due to added security

Performance of some timing tests on the secure transmission and secure device driver are described below. The timing results vary with different encryption algorithms. The tests were performed on a 1Ghz Pentium 4 personal computer by looping over many different calls of the encryption algorithms and using C library function clock to make timing measurements. This method is portable, if not the most refined.



•Figure 9 Performance graph

The figure 9 shows two graphs, the transmission throughput between two endpoints and the throughput using a secure device driver. It is obvious that the proposed secure streaming media model is slower than the generic model but it is so within reasonable limits. The added cost is of scrambling and de-scrambling of packets which is equivalent to solving polynomial equations. The time taken is so constant for scrambling and de-scrambling of data packets. The transmission throughput saturates as we increase the number of octets in the packet. The cryptographic mechanisms used by the server and client will be the bottleneck for performance. Crypto hardware accelerators may be used on both ends to improve performance. An accelerator may also be used to generate key stream for each transaction.

5.2 Comparison of secure RTP, RTP and HTTPS

HTTPS and RTP clients were also implemented for comparing with the proposed secure RTP model. Table 4 contains the startup latency and throughput times for streaming 4003604 bytes. From the results in table 4, it can be seen that the difference in startup latency for the three protocols is negligible. HTTPS transmission has an advantage over the other two protocols for transmission of longer files. HTTPS uses TCP as the transport layer protocol whereas secure RTP and RTP use UDP as the transport layer protocol. Secure RTP allows the usage of either UDP or TCP as the transport layer protocol.

(Milliseconds)	Secure RTP	RTP	HTTPS
Startup time	2423.07	1616.61	2046.29
End receiving	15313.86	23305.93	5353.86

- **Table 4 Comparison of secure RTP, RTP and HTTPS**

The performance penalty issued by secure RTP appears to be within reasonable limits.

5.3 Testing of secure driver

The implementation has been successfully tested using two sound device drivers.

- Intel 810 audio driver
- Crystal Sound Fusion audio driver

For both the device drivers the implementation was straightforward and involved compiling the de-scrambling algorithms file and modifying the write functions. The device driver maintains the initialized de-scrambling key and chosen algorithm in the current state structure.

5.4 Startup latency and throughput

The implementation uses a very simple streaming mode with predefined initial buffer size. As soon as the initial buffer is filled, the receiver starts writing to the audio device driver. Depending on the available processing power and network cards, the size of the initial buffer can be increased or decreased. This implementation assumes that the network card is available to be used at its maximum throughput. The initial buffer size is predetermined by trial and error method and iteration of the program with different parameters.

In practice, several streaming servers and receiver plug-ins are available which have variable settings for startup latency. This fine tuning can be easily established for a required platform.

From the test cases done, it can be concluded that the proposed security model achieves security without a severe performance penalty.

6 Conclusion

In this thesis, a model for increasing the security of streaming media was presented. The approach, which is based on concepts from digital rights management, adds a measure of integrity protection, but is primarily intended to aid in replay preventions.

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.

Future work can implement audio compression techniques to streaming audio. The thesis work can be extended to video streaming using a streaming server. Moreover sophisticated streaming techniques can be utilized to monitor flow of packets between the two ends.

Appendix A: Annotated Bibliography

Stamp, M., (2003). Digital rights management: the technology behind the hype. *Journal of Electronic Commerce Research*, 4, (3).

Explained in detail the outline of a complete digital rights management system. Also provided background information for a secure system.

Doctorow, C., (2003 May). EFF Consensus at Lawyerpoint, Hollywood wants to plug the 'analog hole'. Retrieved August 2003, from http://bpdg.blogs.eff.org/archives/000113.html

Given an overview of expectations from a secure multimedia system to protect multimedia content. Explained the meaning and connotations of the words analog hole.

Kaufman, C., Perlman, R., Speciner, M. (2002). Network Security: Private Communications in a public world. *Prentice Hall.*

Explains in detail different security protocols and handshake messages for key exchange in a networked world. Also provides an overview of issues to be taken into consideration when designing a secure system.

Anderson, R., (2001). Security Engineering: A Guide to Build Dependable Distributed Systems, (20). *John Wiley and Sons*.

Explained digital pay TV system, security concerns related to the same. Process of evolution of a secure digital pays TV system and different attacks issued on them.

Appendix A: Annotated Bibliography (Cont'd)

H323 Definitions (2003). Retrieved August 2003, from http://www.switch.ch/vconf/ws2003/h323_basics_handout.pdf

Explained the terminology used in audio and video conferencing using H.323 protocol stack.

Key definitions (2003). Retrieved August 2003, from http://csrc.nist.gov/publications/nistpubs/8007/node209.html

Explained different scenarios and environment for key distribution and management. Given graphic images of different scenarios help understand the concept of key distribution and management well.

H.235 Security support for multimedia protocol suite (2003). Retrieved August 2003, from http://www.itu.int/ITU-T/

Recommendation for implementation of security in H.323 protocol stack is given. Given detailed description of procedures to be followed at different stages of message exchange.

Secure RTP (2003). Retrieved August 2003, from http://srtp.sourceforge.net

Explained in detail the open source library implementation for secure RTP. Also provided implementation examples of the open source library.

Media-S (2003). Retrieved August 2003, from http://www.sidespace.com/products/medias/

Open source solution of digital rights management system is given. Focused on license management, user privileges and revocation.

APPENDIX A: Annotated Bibliography (Cont'd)

Wheeler, D., Needham, R., (1994). TEA, a tiny encryption algorithm. Retrieved August 2003, from http://www.ftp.cl.cam.ac.uk/ftp/papers/djw-rmn/djw-rmn-tea.html

Explained in detail the design and implementation of tiny encryption algorithm. Simplified code in C language for tiny encryption algorithm can be easily ported.

Schneier, B., (1996). Applied Cryptography, II, *John Wiley and Sons.*

Provided good textbook material to learn and understand cryptography terms and conditions. For those interested in the mathematics of cryptography, also provided the derivation of equations and solutions to several problems.

Windows Media Digital Rights Management Offering (2003). Retrieved August 2003, from http://www.microsoft.com/windows/windowsmedia/wm7/drm/offering.aspx

Explained the features of the product and digital rights management features offered in the same system. Understand the current market offerings by one of the leading players in the market today.

Mishra, P., Stamp, M., (2003). Software uniqueness: how and why. *Proceedings of ICCSA.*

Explained different techniques to generate unique software. Provided an overview of a system essential to establish uniqueness.

Balepin, I., (2003). Retrieved August 2003, from http://wwwcsif.cs.ucdavis.edu/~balepin/new_pubs/worms-cryptovirology.pdf

Explained different techniques to generate cryptographic worms. Provided an overview of a system essential to establish uniqueness.

APPENDIX A: Annotated Bibliography (Cont'd)

Rivest, R., (1992). MD5, Retrieved August 2003, from http://userpages.umbc.edu/~mabzug1/cs/md5/md5.html

Explained MD5 encryption algorithm used in the open SSL library. Definition of the functionality of the algorithm is also given.

Grabbe, J., (2003) DES algorithm illustrated. Retrieved August 2003, from http://www.aci.net/kalliste/des.htm

Explained how DES algorithm works. Brief summary of the history behind DES is also given.

Appendix B: Streaming server web page

Audio Streaming Server

Secure Streaming

Crosby	Play
Drums	Play
Still & Nash	Play
Banjo	Play

HTTPS Streaming

Crosby File

RTP Streaming

Crosby File	Play

MP3 Streaming

Bed rock crowd invasion

Appendix C: Makefiles for compiling different modules

```
# This Makefile has been simplified as much as possible, by
putting all
# generic material, independent of this specific directory,
into
# ../Rules.make. Read that file for details

TOPDIR  := $(shell cd . ;pwd)
include $(TOPDIR)/Rules.make

CFLAGS += -I.. -O

OBJS = cs46xx_secure.o cs46xx_partial.o cs46xx_broken.o
i810_audio_secure.o i810_audio_partial.o i810_audio_broken.o

CFLAGSRTP = -Wall -O4 -fexpensive-optimizations -funroll-
loops
CDEFSRTP  = -DHAVE_CONFIG_H
INCDIR = -I./include/
LIBSRTP   =  -lsrtp
LIBDIRRTP = -L.
LIBDES   =  -lcrypt
LIBPTHREAD = -lpthread

all: $(OBJS) sender sender_rtp secure_receiver
partial_receiver broken_receiver https_receiver rtp_receiver
createclient

cs46xx_secure.o: cs46xx_audio_secure.o secure_tea.o
     $(LD) -r $^ -o $@

cs46xx_partial.o: cs46xx_audio_secure.o partial_tea.o
     $(LD) -r $^ -o $@

cs46xx_broken.o: cs46xx_audio_secure.o broken_tea.o
     $(LD) -r $^ -o $@

i810_audio_secure.o: i810_secure.o secure_tea.o
     $(LD) -r $^ -o $@

i810_audio_partial.o: i810_secure.o partial_tea.o
     $(LD) -r $^ -o $@

i810_audio_broken.o: i810_secure.o broken_tea.o
     $(LD) -r $^ -o $@

message.o: message.c
     $(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@

sender_tea.o: sender_tea.c
     $(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@
```

Appendix C: Makefiles (Cont'd)

```
secure_tea.o: secure_tea.c
	$(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@

partial_tea.o: partial_tea.c
	$(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@

broken_tea.o: broken_tea.c
	$(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@

parseutils.o: parseutils.c
	$(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@

fileutils.o: fileutils.c
	$(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@

stest_secure.o: stest_secure.c
	$(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@

stest_insecure.o: stest_insecure.c
	$(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@

srtp.o: srtp.c
	$(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@

rtp.o: rtp.c
	$(CC) $(CDEFSRTP) -c $(CFLAGSRTP) $(INCDIR) $< -o $@

sender: sender.c sender_tea.o libsrtp.a
	$(CC) $(CDEFSRTP) $(CFLAGSRTP) $(INCDIR) sender_tea.o
$< -o $@ $(LIBDIRRTP) $(LIBSRTP) $(LIBDES)

sender_rtp: sender_rtp.c libsrtp.a
	$(CC) $(CDEFSRTP) $(CFLAGSRTP) $(INCDIR) $< -o $@
$(LIBDIRRTP) $(LIBSRTP) $(LIBDES)

secure_receiver: secure_receiver.c stest_secure.o libsrtp.a
	$(CC) $(CDEFSRTP) $(CFLAGSRTP) $(INCDIR) stest_secure.o
$< -o $@ $(LIBDIRRTP) $(LIBSRTP) $(LIBPTHREAD)
	chmod 4711 $@

partial_receiver: partial_receiver.c stest_secure.o
libsrtp.a
	$(CC) $(CDEFSRTP) $(CFLAGSRTP) $(INCDIR) stest_secure.o
$< -o $@ $(LIBDIRRTP) $(LIBSRTP) $(LIBPTHREAD)
	chmod 4711 $@

broken_receiver: broken_receiver.c stest_secure.o libsrtp.a
	$(CC) $(CDEFSRTP) $(CFLAGSRTP) $(INCDIR) stest_secure.o
$< -o $@ $(LIBDIRRTP) $(LIBSRTP) $(LIBPTHREAD)
	chmod 4711 $@

https_receiver: https_receiver.c stest_insecure.o libsrtp.a
```

```
     $(CC) $(CDEFSRTP) $(CFLAGSRTP) $(INCDIR)
stest_insecure.o $< -o $@ $(LIBDIRRTP) $(LIBSRTP)
$(LIBPTHREAD)
     chmod 4711 $@

rtp_receiver: rtp_receiver.c stest_insecure.o libsrtp.a
     $(CC) $(CDEFSRTP) $(CFLAGSRTP) $(INCDIR)
stest_insecure.o $< -o $@ $(LIBDIRRTP) $(LIBSRTP)
$(LIBPTHREAD)
     chmod 4711 $@

createclient: createclient.c
     $(CC) $< -o $@ $(LIBDES)

parserobj = fileutils.o parseutils.o message.o

srtpobj = srtp.o rtp.o


ciphers = crypto/cipher/cipher.o crypto/cipher/null-cipher.o
\
          crypto/cipher/rijndael-tables.o
\
          crypto/cipher/rijndael.o crypto/cipher/rijndael-
icm.o   \
          crypto/cipher/seal.o

hashes  = crypto/hash/null-auth.o crypto/hash/tmmhv2.o
crypto/hash/sha1.o \
          crypto/hash/auth.o

replay  = crypto/replay/rdb.o crypto/replay/rdbx.o
\
          crypto/replay/ut-sim.o

math    = crypto/math/datatypes.o crypto/math/gf2_8.o
\
          crypto/math/stat.o

ust     = crypto/ust/ust.o

cryptobj = $(ciphers) $(hashes) $(replay) $(math) $(ust)

gdoi    =

libsrtp.a: $(parserobj) $(srtpobj) $(cryptobj) $(gdoi)
     ar cr libsrtp.a $(parserobj) $(srtpobj) $(cryptobj)
$(gdoi)
     ranlib libsrtp.a

install:
     install -d $(INSTALLDIR)
     install -c $(OBJS) $(INSTALLDIR)
```

```
web:
     mkdir -p /home/web/audio
     cd httpd; \
     make;

clean:
     rm -f *.o *~ core
     rm -f secure_receiver partial_receiver broken_receiver
sender createclient

RT=
BUILDDIR=/root/thesis/my/httpd/src
PSTRIP=strip
GZDIR=/root/thesis/my/httpd
TREE=/root/thesis/my/httpd/tree

all: ssl thttpd mycgi
#
# ssl rules
#
# For arm build we cannot simply use config since it has
# no arm settings. Therefore we manipulate the Makefile
# directly.
#
# This make will add header files to /include/openssl
# and it will add libraries to /usr/lib

SSLSRC=$(BUILDDIR)/ssl-src
SSLROOTDIR=$(RT)/usr/local/
SSLSHAREDLIBDIR=$(RT)/lib

ssl:  $(SSLSRC)

     cd $(SSLSRC); \
     ./config --prefix=$(SSLROOTDIR) --
openssldir=$(SSLROOTDIR) -shared ;

     cd $(SSLSRC); \
     make ; \
     make linux-shared; \
     make install ;
     cd $(SSLROOTDIR)/bin ; \
     $(PSTRIP) openssl;
     mkdir -p /include/openssl;
     cd $(SSLROOTDIR)/include; \
     cp -f openssl/* /include/openssl;
     cd $(SSLROOTDIR); \
     rm -Rf man;

     mkdir -p /usr/lib;

# if you want archive libraries
#    mv $(SSLROOTDIR)/lib/libcrypto.a /usr/lib;
```

```
#    mv $(SSLROOTDIR)/lib/libssl.a /usr/lib;
     # we don't want archive libraries so we rm them
     rm -f $(SSLROOTDIR)/lib/libcrypto.a;
     rm -f $(SSLROOTDIR)/lib/libssl.a

     $(PSTRIP) -g $(SSLSRC)/libcrypto.so* ; \
     $(PSTRIP) -g $(SSLSRC)/libssl.so*
     cp -a $(SSLSRC)/libcrypto.so* /usr/lib/. ; \
     cp -a $(SSLSRC)/libssl.so* /usr/lib/. ;
     # if we are using the shared libraries of SSL then
     # load them into the result-root directory
     cd /usr/lib/; \
     cp -a libcrypto.so* $(SSLSHAREDLIBDIR); \
     cp -a libssl.so* $(SSLSHAREDLIBDIR);

$(SSLSRC):
     tar -xvzf $(GZDIR)/openssl*tar.gz -C $(BUILDDIR) ;
     ln -sf $(BUILDDIR)/openssl* $(SSLSRC) ;

ssl-clean:
     rm -rf $(BUILDDIR)/openssl*
     rm -f $(SSLSRC)
     rm -f $(SSLROOTDIR)/openssl.cnf
     rm -f $(SSLROOTDIR)/bin/openssl
     rm -rf /usr/include/openssl

#thttpd

THTTPDSRC=$(BUILDDIR)/thttpd-src

thttpd: $(RT)/usr/sbin/thttpd

$(RT)/usr/sbin/thttpd: $(THTTPDSRC)

     cd $(THTTPDSRC); \
     ln -sf /include/openssl openssl; \
     CC=$(PCC) ./configure --host=i686 --with-ssl;
     cd $(THTTPDSRC) ; \
     patch -N -p1 < $(GZDIR)/thttpd-2.19.patch ; \
     cd $(THTTPDSRC); make thttpd; $(PSTRIP) thttpd ; \
     mkdir -p $(RT)/usr/sbin; cp thttpd $(RT)/usr/sbin ;
     cd $(THTTPDSRC)/extras; make; $(PSTRIP) htpasswd ; \
     mkdir -p $(RT)/usr/sbin; cp htpasswd $(RT)/usr/sbin ;
     cp -Rf $(TREE)/etc $(RT)/etc;
     mkdir -p $(RT)/home/web/audio;
     cp -Rf $(TREE)/home/web $(RT)/home/web;

$(THTTPDSRC):
     tar -xvzf $(GZDIR)/thttpd-*gz -C $(BUILDDIR)
     ln -sf $(BUILDDIR)/thttpd* $(THTTPDSRC)
thttpd-clean:
     rm -rf $(BUILDDIR)/thttpd*
     rm -f $(RT)/usr/sbin/thttpd
mycgi:
```

```
     cd cgi; \
     make;

mycgi-clean:
     cd cgi; \
     make clean;
clean: ssl-clean thttpd-clean mycgi-clean
```

Appendix D: CGI-scripts

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/io.h>
#include <sys/stat.h>
#include <errno.h>
#include <sys/file.h>
#include <fcntl.h>
#include <string.h>

/* script to get user name and ip address of remote
workstation */
int main(int argc, char **argv)
{
  FILE *fp = NULL;
  char str[200],newstr[200],*tmp;
  char *user = getenv("REMOTE_USER");
  printf("content-type:text/html\n\n");
  fp = fopen("/home/web/index.html","r");
  if(fp == NULL) {
    printf("<html><body>Index file not
found</body></html>\n");
    return 0;
  }
  while(fgets(str,199,fp) != NULL)
  {
    if((tmp = strstr(str,"XXXUSER")) == NULL)
      printf("%s\n",str);
    else {
      *tmp = '\0';
      snprintf(newstr,199,"%s%s\">",str,user);
      printf("%s\n",newstr);
    }
  } //end of reading input file
  return 0;
}

/* A simple param file to process parameters passed by get
or post methods for cgi
   written by Deepali Holankar April , 2002
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include "param.h"
```

```
/* Function Definitions */
/* Parse all http post parameters and store them in hash
table */
CParam::CParam(void)
{
  symTab=NULL;
  glInitialized =0;
  ptrSymTab = NULL;
  int ret =
ParseAllCGIInput(GetAllCGIInput(GetContent_Length()));
}

/* Get first value of given http post parameter*/
char * CParam::GetFirstValue(char *key)
{
  ptrSymTab = symTab;
  return (Search(key));
}
/* Get next value of given http post parameter */
char * CParam::GetNextValue(char *key)
{
  if (!ptrSymTab) return NULL;
  ptrSymTab = ptrSymTab->next;

  return (Search(key));
}

char * CParam::Search(char *key)
/* search for the next match in symbol table */
{
  while (ptrSymTab) {
    if (strcasecmp(key, ptrSymTab->key) == 0)
      return ptrSymTab->value;
    ptrSymTab = ptrSymTab->next;
  }
  return NULL;
}

CParam::~CParam()
{
  SymtabEntry *p, *old;

  p = symTab;
  while (p) {
    free(p->key);
    free(p->value);
    old = p;
    p = p->next;
    free(old);
  }
  glInitialized = 0;
}
```

```
int CParam::GetContent_Length(void)
{
  char *value;
  int length;

  value = getenv("CONTENT_LENGTH");
  if (!value) return -1;
  if (sscanf(value, "%d", &length) != 1) return -1;

  return length;
}

char * CParam::GetAllCGIInput(int length)
{
  char *input, *query_string;

  if (length < 0) {
    /* get input from environment variable "QUERY_STRING" */
    query_string = getenv("QUERY_STRING");
    if (!query_string) return NULL;
    input = (char *) malloc(sizeof(char) *
(strlen(query_string) + 1));
    if (input != NULL)
      strcpy(input, query_string);
  }
  else {
    /* get input from stdin */
    input = (char *) malloc (sizeof(char) * (length + 1));
    if (input != NULL) {
      fgets(input, length+1, stdin);
      input[length] = '\0';
    }
  }
  if (!input)
    fprintf(stderr, "Out of memory.\n");
  return input;
}

int CParam::ParseAllCGIInput(char *input)
{
  char *startKey, *startVal, *s;
  int index, keyLen, valLen, done = 0;
  SymtabEntry *symTabTail, *newSym;

  if (!input) return 0;
  symTabTail = symTab;

  s = input;
  while (!done) {
    startKey = s;  keyLen = 0;

    /* look for '=' */
   while ((*s) != '=' && (*s) !='\0') {
```

```
      s++; keyLen++;
    }
    if ((*s) == '\0') {
      /* incomplete input string at the end */
      done = 1;
      break;
    }
    (*(s++)) = '\0';

    startVal = s; valLen = 0;

    /* look for '&' */
    while ((*s) != '&' && (*s) !='\0') {
      s++; valLen++;
    }
    if ((*s) == '\0')
      done = 1;   /* this is the last entry */
    (*(s++)) = '\0';

    /* allocate space for the new symbol */
    newSym = (SymtabEntry *) malloc (sizeof(SymtabEntry));
    if (!newSym) {
      fprintf(stderr, "Out of memory.\n");
      break;
    }
    newSym->next = NULL;
    newSym->key = (char *) malloc (sizeof(char) * (keyLen +
1));
    if(valLen > 0)
      newSym->value = (char *) malloc (sizeof(char) *
(valLen + 1));
    else
      newSym->value = NULL;
    if (!(newSym->key) || (valLen > 0 && !(newSym->value)))
{
      fprintf(stderr, "Out of memory.\n");
      break;
    }
    CopyCGIString(newSym->key, startKey);
    CopyCGIString(newSym->value, startVal);

    /* append the new entry to symTab */
    if (!symTabTail)
      symTab = newSym;
    else
      symTabTail->next = newSym;
    symTabTail = newSym;
  }
  free(input);
  glInitialized = 1;
  return 1;


}
```

```
void CParam::CopyCGIString(char *dest, char *src)
{
  if(src == NULL)
  {
    dest = NULL;
    return;
  }
  if(strlen(src) <= 0)
  {
      dest = NULL;
      return;
  }
  char *d, *s, a, b, c;
  d = dest;  s = src;
  char *tmpstr = (char
*)malloc(sizeof(char)*(strlen(src)+1));
  d=tmpstr;
  while (*s) {
    c = *s;
    if (c == '+')   /* plus is a space */
      c = ' ';
    else if (c == '%') {  /* convert characters */
      a = *(++s);
      if (a=='\0') break;  /* this shouldn't happen */
      b = *(++s);
      if (b=='\0') break;  /* this shouldn't happen */
      a = a - ((a >= '0' && a <= '9') ? ('0') : ('A' - 10));
      b = b - ((b >= '0' && b <= '9') ? ('0') : ('A' - 10));
      c = a * 16 + b;
    }

    (*(d++)) = c;
    s++;
  }
  (*d) = '\0';

  for(d=tmpstr;*d == ' '|| *d == '\r' || *d == '\n' || *d ==
'\t' || *d == '"';d++);
  strcpy(dest,d);
  for(d=dest+strlen(dest)-1; d >= dest ;d--)
  {
    if(*d == ' '||*d == '\r' || *d == '\n' || *d == '\t' ||
*d == '"')
      *d = '\0';
    else
      break;
  }
  free(tmpstr);
}


#ifndef _CFGPARSER_PARAM_H_
#define _CFGPARSER_PARAM_H_
```

```
typedef struct _SymtabEntry {
  char *key;       //htmlname
  char *value;     //current htmlvalue
  struct _SymtabEntry *next;
} SymtabEntry;

class CParam
{
  protected:
  char *Search(char *key);
  int GetContent_Length(void);
  char *GetAllCGIInput(int length);
  int ParseAllCGIInput(char *input);
  void CopyCGIString(char *dest,char *src);

  public:
  CParam(void);
  virtual ~CParam();
  SymtabEntry *symTab;
  int glInitialized;
  SymtabEntry *ptrSymTab;
  char *GetFirstValue(char *key);
  char *GetNextValue(char *key);


};



#endif

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/io.h>
#include <sys/stat.h>
#include <errno.h>
#include <sys/file.h>
#include <fcntl.h>
#include <string.h>
#include "param.h"

#define SECURERECEIVERPORT 13000
#define INSECURERECEIVERPORT 14000


#define MAXPARAMLEN  5000

int becomeroot() {
   int rc = 0;
```

```
   rc = setuid (0);
    if (0 != rc) {
        printf ("<html><body>setuid
%s\n</body></html>",strerror(errno));
        exit (0);
    }
    rc = seteuid (0);
    if (0 != rc) {
        printf ("<html><body>seteuid
%s\n</body></html>",strerror(errno));
        exit(0);
    }
    return rc;
}

int fork_n_execute (const char * cmd0)
{
    pid_t child,granc;
    int   rc=0;

    if (NULL == cmd0)
    {
        return -1;
    }
    char m_cmd[MAXPARAMLEN+1];
    sprintf(m_cmd,"echo `date` %s >> /var/log/ssm\0",cmd0);
    system(m_cmd);

    child = fork();
    if (child < 0)
    {
        system(cmd0);
        return 0;
    }
    else if (0 == child)
    {
        // child
        granc = fork();
        if(granc < 0)
        {
          execl("/bin/bash", "bash", "-c",cmd0, (char *) 0);
          _exit(0);       //   execl error
        }
        else if (granc == 0)
        {
          execl("/bin/bash", "bash", "-c",cmd0, (char *) 0);
          _exit(0);       //
        }
        _exit(0);
    }
    return (rc);
}
int main(int argc, char **argv)
{
```

```
  char newstr[MAXPARAMLEN+1];
  char *tmpstr,*audiofile,*user,*raddr;
  CParam m_par;
  int sampling;
  int secure = 0;

  raddr = getenv("REMOTE_ADDR");

  printf("content-type:text/html\n\n");
  printf("<html><head></head><body>");
  printf("<table>\n");
  tmpstr = raddr;
  while(tmpstr != NULL && *tmpstr != '\0')
  {
    if(*tmpstr == ':')
      raddr = tmpstr;
    tmpstr++;
  }
  if(*raddr == ':')
    raddr++;
  secure = atoi(m_par.GetFirstValue("secure"));
  user = m_par.GetFirstValue("user");
  audiofile = m_par.GetFirstValue("file");
  sampling = atoi(m_par.GetFirstValue("rate"));
  printf("<tr><td>Streaming audio for %s
%s</td></tr>",user,raddr);
  if(secure)
    snprintf(newstr,MAXPARAMLEN,"/root/thesis/my/sender %s
%d /home/web/audio/%s %s
%d",raddr,SECURERECEIVERPORT,audiofile,user,sampling);
  else
    snprintf(newstr,MAXPARAMLEN,"/root/thesis/my/sender_rtp
%s %d /home/web/audio/%s %s
%d",raddr,INSECURERECEIVERPORT,audiofile,user,sampling);
  printf("<tr><td>invoking %s</td></tr>",newstr);
  printf("</table></body></html>\n");
  becomeroot();
  fork_n_execute(newstr);
  return 0;
}
```

Appendix E: Receiver and sender components

```
/*
 * secure_receiver
 *
 *
 */


#include <stdio.h>          /* for printf, fprintf */
#include <stdlib.h>         /* for atoi()          */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>     /* openbsd wants this  */
#include <arpa/inet.h>
#include <errno.h>
#include <unistd.h>         /* for close()         */
#include <string.h>         /* for strncpy()       */
#include <time.h>       /* for usleep()        */
#include <pthread.h>

#include "srtp.h"
#include "rtp.h"

#include "fileutils.h"
#include "parseutils.h"
#include "defaultvalues.h"
#include "message.h"
#include "secure_driver.h"

/* algorithms supported are  5 6 7 9 10 */
#define SECRETDATA
"$1$e0ceb53c$EY7.VRngdVer4bTicm74V1\n$1$e0ceb53c$rmzxJTJXLaa
AJHOS2BCaM/\n$1$e0ceb53c$pCJotupJvvNVVPvmj1keU0\n$1$e0ceb53c
$kTAvySIymtpvZGB4kLBHW1\n$1$e0ceb53c$/Us4XCnNuYhM.RP9/spnj0\
n"

#define MYKEY
"a2ee93717da76195bb878578790af71c4ee9f859e197a414a78d5abc745
1"

#define TIMEOUT_SECONDS  30
#define ADDR_IS_MULTICAST(a) IN_MULTICAST(htonl(a))

struct security_info secureinfo;
/*
 * usage prints an error message describing how this program
should be
 * called, then calls exit()
 */

void
usage(char *prog_name);
```

```
/*
 * leave_group(...) de-registers from a multicast group
 */

void
leave_group(int sock, struct ip_mreq mreq, char *name);


/*
 * program_type distinguishes the [s]rtp sender and receiver
cases
 */

typedef enum { sender, receiver, unknown } program_type;


/* read the personal key from MYKEY_FILE */
int
get_my_key(unsigned char **mykey);

int
receiverthread(void);

message_t *receivedmsgs = NULL;
int samplingrate=11000;
int packets = 0;
extern int audio_fd;
int wrote_ptr=0,ctr =0,wctr=0;
struct timeval timeout;
fd_set fdread;
rtp_receiver_t rcvr;
int sock,audio_wfifo,audio_rfifo;
extern int chunk;
extern unsigned char *filbuf;
extern int in_ptr ;
extern int out_ptr;
extern int run_out;
int filesize = 0;
int memindex = 0;
int givesignal = 0;

pthread_mutex_t condition_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condition_cond = PTHREAD_COND_INITIALIZER;

mytime_t mytime;

int
main (int argc, char *argv[]) {
  //struct stat buf;
  double timeElapsed;


  unsigned char word[2*PACKET_SIZE];
  int ret;
```

```
  struct in_addr rcvr_addr;
  struct sockaddr_in name,sendername;
  program_type prog_type = unknown;
  sec_serv_t sec_servs = sec_serv_none;
  struct ip_mreq mreq;
  unsigned char *input_key = NULL;
  //unsigned char *address = NULL;
  unsigned short port = 0;
  rtp_sender_t snd;
  message_t msg;
    int len;
    int sresult,startdevice = 0;
    int sessionflag = 0;
 pthread_t rd,wd;
 pthread_attr_t rattr,wattr;
 //struct sched_param sp;
#if BEW
  struct sockaddr_in local;
#endif
/*BEW */

  pthread_attr_init(&rattr);
  pthread_attr_setschedpolicy(&rattr,SCHED_RR);
  //memset(&sp, 0, sizeof(struct sched_param));
  //sp.sched_priority = sched_get_priority_max(SCHED_RR);
  //pthread_attr_setschedparam(&rattr, &sp);

  pthread_attr_init(&wattr);
  pthread_attr_setschedpolicy(&wattr,SCHED_RR);
  //memset(&sp, 0, sizeof(struct sched_param));
  //sp.sched_priority = sched_get_priority_min(SCHED_RR);
  //pthread_attr_setschedparam(&wattr, &sp);

  sec_servs |= sec_serv_conf;
  sec_servs |= sec_serv_auth;

  prog_type = receiver;

  get_my_key(&input_key);

  if ((sec_servs && !input_key) || (!sec_servs &&
input_key)) {
    /*
     * a key must be provided if and only if security
services have
     * been requested
     */
    printf("input_key not available\n");
    usage(argv[0]);
  }


  printf("security services: ");
  if (sec_servs & sec_serv_conf)
```

```
    printf("confidentiality ");
  if (sec_servs & sec_serv_auth)
    printf("message authentication");
  if (sec_servs == sec_serv_none)
    printf("none");
  printf("\n");

  if (argc < 2 || argc > 3) {
    /* wrong number of arguments */
    usage(argv[0]);
  }

  /* set port from arg */
  port = atoi(argv[1]);

  /* open socket */
  sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
  if (sock < 0) {
    fprintf(stderr, "%s: couldn't open socket\n", argv[0]);
    exit(1);
  }

  name.sin_addr.s_addr   = htonl(INADDR_ANY);
  name.sin_family = AF_INET;
  name.sin_port   = htons(port);

  if (ADDR_IS_MULTICAST(rcvr_addr.s_addr)) {

    mreq.imr_multiaddr.s_addr = rcvr_addr.s_addr;
    mreq.imr_interface.s_addr = htonl(INADDR_ANY);
    ret = setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
&mreq, sizeof(mreq));
    if (ret < 0) {
      fprintf(stderr, "%s: Failed to join multicast group",
argv[0]);
      perror("");
      exit(1);
    }
  }


    if (bind(sock, (struct sockaddr *)&name, sizeof(name)) <
0) {
      close(sock);
      fprintf(stderr, "%s: socket bind error\n", argv[0]);
      perror(NULL);
      if (ADDR_IS_MULTICAST(rcvr_addr.s_addr)) {
     leave_group(sock, mreq, argv[0]);
      }
      exit(1);


    }
```

```
    chunk = 1 << FRAG_SIZE ;

    printf("chunk %d\n",chunk);

    rtp_receiver_init(&rcvr, sock, name);
    srtp_receiver_init(&rcvr, name, sec_servs, input_key);
    sessionflag = 0;
    packets = 0;
    ctr = 0;
    wctr = 0;
    memindex = 0;
    while (sessionflag == 0 || ctr < packets)
    {
        timeout.tv_sec = TIMEOUT_SECONDS;
        timeout.tv_usec = 0;
        FD_ZERO(&fdread);
        FD_SET(sock, &fdread);
        len = 2*PACKET_SIZE;
        message_init(&msg,0,0);

        if ((sresult = select(sock + 1, &fdread, NULL, NULL,
&timeout)) < 0)
        {
                        printf("Error: select() failed,
errno <%d>\n", errno);
                        close(sock);
                        exit(2);
        }
        if (sresult != 0) // got data
        {
                        if(FD_ISSET(sock,&fdread)) {
                           if (rtp_recvfrom(&rcvr,word,&len)
> -1) {
                                memcpy((void *)&msg,(void
*)word,sizeof(message_t));
                                if(sessionflag == 0) {
                                   printf("\n\tgot session
key %d\n",msg.msg_hdr.seq_num);
                                   packets =
msg.msg_hdr.seq_num; /* no of packets */
                                   filesize =
msg.msg_hdr.length;
                                   /* send the supported
algorithms info */
                          bzero(&sendername,
sizeof(sendername));
                                   sendername.sin_family =
AF_INET;
                       my_awkstr(msg.data,"
",5,word,PACKET_SIZE);


                                   samplingrate =
atoi(word);
```

```
                        my_awkstr(msg.data,"
",4,word,PACKET_SIZE);
                                    memcpy((void
*)secureinfo.teakey,(void *)word,16);
                        my_awkstr(msg.data,"
",3,word,PACKET_SIZE);
                                    sendername.sin_port =
htons(atoi(word));
                        my_awkstr(msg.data,"
",2,word,PACKET_SIZE);

sendername.sin_addr.s_addr = inet_addr(word);
                                    printf("\n\tsender ip %s
%d\n",inet_ntoa(sendername.sin_addr),ntohs(sendername.sin_po
rt));
                        my_awkstr(msg.data,"
",1,word,PACKET_SIZE);
                                    rtp_sender_init(&snd,
sock, sendername);
                                    srtp_sender_init(&snd,
sendername, sec_servs,word);
                                    srtp_receiver_init(&rcvr,
name, sec_servs, word);

strncpy(word,SECRETDATA,PACKET_SIZE);
                                    sresult = strlen(word);

message_init(&msg,MSG_TYPE_ALGO,sresult);
                                    memcpy((void
*)msg.data,(void *)word,sresult);
                                    printf("Sending algo
supported\n");

rtp_sendto(&snd,&msg,sizeof(message_t));
                                    sessionflag = 1;
                                    receivedmsgs = (message_t
*)malloc(sizeof(message_t) * packets);

bzero(receivedmsgs,sizeof(message_t)*packets);
                                    filbuf = (char *)
malloc(sizeof(char) * filesize);

bzero(filbuf,sizeof(char)*filesize);
                                    startdevice = STARTDEVICE
;
                                    /* if(samplingrate >
43000)
                                         startdevice =
(FILE_FRAGS *chunk * 16);



                                    else if(samplingrate >
21000)
```

```
                                            startdevice =
(FILE_FRAGS *chunk * 7);
                                       else
                                            startdevice =
(FILE_FRAGS *chunk) + chunk;
                                       */
                                    } else {
                                       if(msg.msg_hdr.msg_type
== MSG_TYPE_ALGO){

                                           secureinfo.algo =
msg.msg_hdr.seq_num;

                                           printf("Scrambling
algo %d\n",secureinfo.algo);
                                       } else {
                                         printf("Received packet
%d seqnum %d\n",ctr,msg.msg_hdr.seq_num);
                                         if(msg.msg_hdr.seq_num
< 0 || msg.msg_hdr.seq_num >= packets) {
                                               printf("Got
corrupt packet\n");

                                               close(sock);
                                               return 0;
                                         }
                                         if(ctr == 0) {

gettimeofday(&(mytime.startTime), NULL);
                                         }
                                         ctr++;
                                         memcpy((void
*)&(receivedmsgs[msg.msg_hdr.seq_num]),(void
*)&msg,sizeof(message_t));
                                         while(wctr <
msg.msg_hdr.seq_num+1) {

if(receivedmsgs[wctr].msg_hdr.length <= 0)
                                                   break;

memcpy(&filbuf[memindex],receivedmsgs[wctr].data,receivedmsg
s[wctr].msg_hdr.length);
                                            memindex = memindex
+ receivedmsgs[wctr].msg_hdr.length;
                                            in_ptr = memindex /
chunk;

                                            wctr++;
                                         }
                                       }
                                       if(memindex >=
startdevice) {

gettimeofday(&(mytime.audioTime), NULL);


                                          printf("Done
prefill\n");
```

```
                                        printf("create reader
thread\n");
                                        sresult =
pthread_create(&rd,&rattr, (void *)receiverthread, NULL);
                                        if( sresult != 0 ) {
                                          perror("  Can't
create reader thread... ") ;
                                          close(sock);
                                        }
                                        printf("create writer
thread\n");
                                        sresult =
pthread_create(&wd, &wattr, (void *)stest_main, NULL);
                                        if( sresult != 0 ) {
                                          perror("  Can't
create writer thread... ") ;
                                          close(sock);
                                        }
                                        sessionflag = 2;
                                        break;
                                     }
                                  }
                             }
                             else {
                                  printf("\terror while
receiving bytes received %d\n",len);
                                  close(sock);
                                  return 0;
                             }
                          }
        }
        else //timedout
        {
                          if(sessionflag != 0) {
                            printf("Receiver timedout\n");
                            close(sock);
                            return 0;
                          }
        }
     } /* end of while  */
if(sessionflag == 2) {
   pthread_join(rd,NULL);
   pthread_join(wd,NULL);
} else
   stest_main();

  /*
   printf("writing to sound\n");
   stest_main();
   printf("writing from receivedmsgs\n");


   for(sresult = 0; sresult < packets ; sresult++) {
```

```
write(audio_fd,receivedmsgs[sresult].data,receivedmsgs[sresu
lt].msg_hdr.length);
    }
  {
   FILE *fp = NULL;
   fp = fopen("soundfile-msgs","w");
   for(ctr = 0; ctr < packets; ctr++)

fwrite(receivedmsgs[ctr].data,1,receivedmsgs[ctr].msg_hdr.le
ngth,fp);
   fclose(fp);
   fp = fopen("soundfile-buf","w");
   fwrite(filbuf,1,filesize,fp);
   fclose(fp);
  }
   */
   free(receivedmsgs);
   if (ADDR_IS_MULTICAST(rcvr_addr.s_addr)) {
     leave_group(sock, mreq, argv[0]);
   }
   close(sock);

  timeElapsed = mytime.audioTime.tv_sec * 1000000 +
mytime.audioTime.tv_usec -
                        mytime.startTime.tv_sec * 1000000 +
mytime.startTime.tv_usec;
  printf("Startup Time: <%.2f> ms\n", timeElapsed/1000);
  timeElapsed = mytime.endTime.tv_sec * 1000000 +
mytime.endTime.tv_usec -
                        mytime.startTime.tv_sec * 1000000 +
mytime.startTime.tv_usec;
  printf("End receiving: <%.2f> ms\n", timeElapsed/1000);
  return 0;
}


int receiverthread(void) {
  FILE *fp = NULL;
  unsigned char word[2*PACKET_SIZE];
  message_t msg;
  int len,sresult;

  message_init(&msg,0,0);
  while(ctr < packets) {
        timeout.tv_sec = TIMEOUT_SECONDS;
        timeout.tv_usec = 0;
        FD_ZERO(&fdread);
        FD_SET(sock, &fdread);
        len = 2*PACKET_SIZE;
```

```
        if ((sresult = select(sock + 1, &fdread, NULL, NULL,
&timeout)) < 0)
        {
                        printf("Error: select() failed,
errno <%d>\n", errno);
                        exit(2);
        }
        if (sresult != 0) // got data
        {
                        if(FD_ISSET(sock,&fdread)) {
                           if (rtp_recvfrom(&rcvr,word,&len)
> -1) {
                                     memcpy((void
*)&msg,(void *)word,sizeof(message_t));

                                     if(msg.msg_hdr.seq_num
< 0 || msg.msg_hdr.seq_num >= packets) {
                                           printf("Got
corrupt packet\n");
                                           return 0;
                                     }
                                     memcpy((void
*)&(receivedmsgs[msg.msg_hdr.seq_num]),(void
*)&msg,sizeof(message_t));
                                     ctr++;
                                     /* loop to verify we
dont get packet 10 before packet 8 or 9 */
                                     while(wctr <
msg.msg_hdr.seq_num+1) {

if(receivedmsgs[wctr].msg_hdr.length <= 0)
                                               break;

memcpy(&filbuf[memindex],receivedmsgs[wctr].data,receivedmsg
s[wctr].msg_hdr.length);
                                        memindex = memindex
+ receivedmsgs[wctr].msg_hdr.length;
                                        wctr++;
                                     }
                                     if(givesignal == 1 &&
                                           ((out_ptr +
(FILE_FRAGS * chunk)+chunk) < memindex || memindex ==
filesize)) {
                                           printf("giving
signal %d\n",memindex);

pthread_mutex_lock(&condition_mutex);

pthread_cond_signal(&condition_cond);
                                           givesignal = 0;

pthread_mutex_unlock(&condition_mutex);
```

```
                                    }
                            }
                        }
        }
        else //timedout
        {
                          printf("Receiver timedout\n");
                          break;
        }

  }
  gettimeofday(&(mytime.endTime), NULL);
  run_out = 1;
  pthread_mutex_lock(&condition_mutex);
  pthread_cond_signal(&condition_cond);
  pthread_mutex_unlock(&condition_mutex);
  fp = fopen("soundfile","w");
  for(ctr = 0; ctr < packets; ctr++)

fwrite(receivedmsgs[ctr].data,1,receivedmsgs[ctr].msg_hdr.le
ngth,fp);
  fclose(fp);
  return 0;
}

void
usage(char *string) {

  printf("usage: %s receivingport [startdevice]\n",
      string);
  exit(1);

}


void
leave_group(int sock, struct ip_mreq mreq, char *name) {
  int ret;

  ret = setsockopt(sock, IPPROTO_IP, IP_DROP_MEMBERSHIP,
&mreq, sizeof(mreq));
  if (ret < 0) {
     fprintf(stderr, "%s: Failed to leave multicast group",
name);
     perror("");
  }
}



/* read the personal key from MYKEY_FILE */
int get_my_key(unsigned char **mykey){
```

```
  *mykey = NULL;
  *mykey = (unsigned char *)strdup(MYKEY);
  return 0;
}


   /*
  char *keysfile = MYKEY_FILE;
  FILE *fp = NULL;
  if(initscanner(keysfile,&fp) != 0)
     return -1; // cannot open keys file

  while(getscannerdata(&fp,data,MAXFILESTR) == 0) {
         right_left_trim(data);
         *mykey = (unsigned char *)strdup(data);
         break;
  }
  deinitscanner(&fp);
     printf("Decoding packets with teakey %s\n",tea_key);
     for(ctr = 0; ctr < packets; ctr++) {
      int i =0;
      memcpy((void *)&msg,(void
*)&receivedmsgs[ctr],sizeof(message_t));
     printf("received data %d of length %d\n",ctr,i);
     receiver_print_hex(msg.data,i);
     for(i = 0; i < PACKET_SIZE/(2 *sizeof(long));i++) {
         receiver_decode(algo,(long
*)&(msg.data[2*i*sizeof(long)]),(long *)tea_key,(long
*)&(receivedmsgs[ctr].data[2*i*sizeof(long)]));
     }
     i = receivedmsgs[ctr].msg_hdr.length;
     receivedmsgs[ctr].data[i]= '\0';
     printf("\noriginal\n");
     receiver_print_hex(receivedmsgs[ctr].data,i);
     printf("\n");
   }
   end of decoding */

/*
 * sender.c
 *
 * srtp packets sender
 * Usage: sender destip destport filename username
 */

#include <stdio.h>          /* for printf, fprintf */
#include <stdlib.h>         /* for atoi()          */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>     /* openbsd wants this  */
#include <arpa/inet.h>
#include <errno.h>
```

```
#include <unistd.h>         /* for close()         */
#include <string.h>         /* for strncpy()       */
#include <time.h>       /* for usleep()        */

#include "srtp.h"
#include "rtp.h"

#include "message.h"
#include "fileutils.h"
#include "parseutils.h"
#include "defaultvalues.h"
#include <openssl/des.h>

#define _XOPEN_SOURCE
#define MAX_SUPPORTEDALGOS  16
#define SERVERKEY "$1$e0ceb53c"
#define KEYS_FILE        "/home/keys.txt"
#define PERMISSIONS_FILE "/home/permissions.txt"
#define BROKEN_FILE      "/home/broken.txt"
#define TIMEOUT_SECONDS  10
#define USEC_RATE        (1)
#define ADDR_IS_MULTICAST(a) IN_MULTICAST(htonl(a))

#define SENDER_DEBUG 1
#define BEW 1
/*
 * usage prints an error message describing how this program
should be
 * called, then calls exit()
 */

void
usage(char *prog_name);

/*
 * leave_group(...) de-registers from a multicast group
 */

void
leave_group(int sock, struct ip_mreq mreq, char *name);


/*
 * program_type distinguishes the [s]rtp sender and receiver
cases
 */

typedef enum { sender, receiver, unknown } program_type;


/* do_permissions will check from the simple permissions.txt
file in the running dir
```

```
   if a given user has permissions to request the file or
not
*/

int
do_permissions(const char *username,const char *filename);

/* read the key from KEYS_FILE */
int
get_client_key(const char *username,unsigned char
**clientkey);

/* generate session key */
int
get_session_key(unsigned char **mykey,unsigned char
*teakey);

/* select scrambling algorithm */
int
select_scrambling_algo(int *clientalgo,int *serveralgo,const
char *supportedalgos,unsigned char *sessionkey);

int
main (int argc, char *argv[]) {
  char *dictfile = NULL;
  FILE *dict;
  message_t msg;
  int sock, ret;
  struct in_addr rcvr_addr;
  struct sockaddr_in name;
  program_type prog_type = unknown;
  sec_serv_t sec_servs = sec_serv_none;
  unsigned char ttl = 5;
  struct ip_mreq mreq;
  int len,ctr,sresult;
  unsigned char word[PACKET_SIZE+1];
  unsigned char myip[20];
  unsigned char *input_key = NULL;
  unsigned char tea_key[17];
  unsigned char *address = NULL;
  unsigned short port = 0,srvport = 0;
  rtp_sender_t snd;
  rtp_receiver_t rcvr;
  struct timeval timeout;
  fd_set fdread;
  char *username = NULL;
  struct stat buf;
  int clientalgo,serveralgo;
  int samplingrate = 44100;
  struct timespec ts;
#if BEW
  struct sockaddr_in local;
```

```
#endif
/*BEW */

  /* check args */
  if ( 6 != argc) {
    /* wrong number of arguments */
    usage(argv[0]);
  }
  prog_type = sender;

  sec_servs |= sec_serv_conf;
  sec_servs |= sec_serv_auth;

  username = argv[4];

  if(username == NULL || strlen(username) <= 0) {
    usage(argv[0]);
  }
  get_client_key((const char *)username,&input_key);

  if ((sec_servs && !input_key) || (!sec_servs &&
input_key)) {
    /*
     * a key must be provided if and only if security
services have
     * been requested
     */
    usage(argv[0]);
  }

  samplingrate = atoi(argv[5]);
  printf("security services: ");
  if (sec_servs & sec_serv_conf)
    printf("confidentiality ");
  if (sec_servs & sec_serv_auth)
    printf("message authentication");
  if (sec_servs == sec_serv_none)
    printf("none");
  printf("\n");


  /* set address from arg */
  address = argv[1];

  /* set port from arg */
  port = atoi(argv[2]);

  /* set requested filename */
  dictfile = (char *) argv[3];

  if(do_permissions(username,dictfile) == 0) {
```

```
       printf("receiver user %s does not have permissions on
file %s\n",username,dictfile);
       exit(1);
  }

#if HAVE_INET_ATON
  if (0 == inet_aton(address, &rcvr_addr)) {
    fprintf(stderr, "%s: cannot parse IP v4 address %s\n",
argv[0], address);
    exit(1);
  }
  if (rcvr_addr.s_addr == INADDR_NONE) {
    fprintf(stderr, "%s: address error", argv[0]);
    exit(1);
  }
#else
  rcvr_addr.s_addr = inet_addr(address);
  if (0xffffffff == rcvr_addr.s_addr) {
    fprintf(stderr, "%s: cannot parse IP v4 address %s\n",
argv[0], address);
    exit(1);
  }
#endif

  /* open socket */
  sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
  if (sock < 0) {
    fprintf(stderr, "%s: couldn't open socket\n", argv[0]);
    exit(1);
  }

  name.sin_addr   = rcvr_addr;
  name.sin_family = AF_INET;
  name.sin_port   = htons(port);

  if (ADDR_IS_MULTICAST(rcvr_addr.s_addr)) {
      ret = setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL,
&ttl,
                    sizeof(ttl));
      if (ret < 0) {
     fprintf(stderr, "%s: Failed to set TTL for multicast
group", argv[0]);
     perror("");
     exit(1);
      }

    mreq.imr_multiaddr.s_addr = rcvr_addr.s_addr;
    mreq.imr_interface.s_addr = htonl(INADDR_ANY);
    ret = setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
&mreq, sizeof(mreq));
    if (ret < 0) {
```

```
      fprintf(stderr, "%s: Failed to join multicast group",
argv[0]);
      perror("");
      exit(1);
    }
  }


#if BEW
    /* bind to local socket (to match crypto policy, if need
be) */
    memset(&local, 0, sizeof(struct sockaddr_in));
    local.sin_addr.s_addr = htonl(INADDR_ANY);
    local.sin_port = htons(0);  //ephemeral port
    ret = bind(sock, (struct sockaddr *) &local,
sizeof(struct sockaddr_in));
    if (ret < 0) {
      fprintf(stderr, "%s: bind failed\n", argv[0]);
      perror("");
      exit(1);
    }
    srvport = sizeof(struct sockaddr_in);
    getsockname(sock,(struct sockaddr *) &local,(socklen_t
*) &srvport); //srvport pointing just as extra length
variable
    srvport = ntohs(local.sin_port);
    local.sin_addr.s_addr   = htonl(INADDR_ANY);
    local.sin_family = AF_INET;
#endif
/*BEW */

    rtp_sender_init(&snd, sock, name);
    srtp_sender_init(&snd, name, sec_servs, input_key);

    /* check the file to be sent */

    if(stat(dictfile,&buf) != 0 || buf.st_size <= 0) {
      fprintf(stderr, "%s: file %s does not exist, or is of
size 0 length\n", argv[0], dictfile);
      if (ADDR_IS_MULTICAST(rcvr_addr.s_addr)) {
     leave_group(sock, mreq, argv[0]);
      }
      exit(1);
    }

    /* open file to be sent */
    dict = fopen (dictfile, "r");
    if (dict == NULL) {
      fprintf(stderr, "%s: couldn't open file %s\n",
argv[0], dictfile);
      if (ADDR_IS_MULTICAST(rcvr_addr.s_addr)) {
     leave_group(sock, mreq, argv[0]);
```

```
      }
      exit(1);
    }

    /* send the session key */
       fprintf(stderr,"generating session keys\n");
       get_session_key(&input_key,tea_key);
       if(!input_key) {
         fprintf(stderr,"could not generate session key\n");
         if (ADDR_IS_MULTICAST(rcvr_addr.s_addr)) {
         leave_group(sock, mreq, argv[0]);
         }
         exit(1);
       }
       fprintf(stderr,"doing message init\n");
       message_init(&msg,MSG_TYPE_INIT,buf.st_size);
       fprintf(stderr,"doing memcpy\n");
       memcpy((void *)msg.data,(void
*)input_key,strlen(input_key)+1);
       fprintf(stderr,"calc total packets \n");
       msg.msg_hdr.seq_num = ((buf.st_size % PACKET_SIZE) ==
0 ? (buf.st_size /PACKET_SIZE) : (buf.st_size/PACKET_SIZE)+1
);
       fprintf(stderr,"total packets=
%d\n",msg.msg_hdr.seq_num);

       fprintf(stderr,"get ip info\n");
       create_tmp_file(word,PACKET_SIZE);
       my_cmd("/home/get_ipinfo.sh eth0 %s",word);
       my_awk(word," ",1,myip,19);
       snprintf(msg.data,MAXFILESTR,"%s %s %d %s
%d",input_key,myip,srvport,tea_key,samplingrate);
       fprintf(stderr,"secretmsg %s\n",msg.data);
       rtp_sendto(&snd,(void *)&msg,sizeof(message_t));

       if(SENDER_DEBUG)
          fprintf(stderr,"sent bytes
%d\n",sizeof(message_t));
       /* change from client key to session key */
       srtp_sender_init(&snd,name,sec_servs,input_key);

    /* receive the algorithms supported */
     {
       printf("Waiting to receive supported algorithms\n");
       rtp_receiver_init(&rcvr, sock, local);
       srtp_receiver_init(&rcvr,local, sec_servs,
input_key);
       timeout.tv_sec = 5*TIMEOUT_SECONDS;
       timeout.tv_usec = 0;
       FD_ZERO(&fdread);
       FD_SET(sock, &fdread);
```

```
       if ((sresult = select(sock + 1, &fdread, NULL, NULL,
NULL)) < 0)
       {
                        printf("Error: select() failed,
errno <%d>\n", errno);
                        close(sock);
                        exit(2);
       }
       if (sresult != 0) // got data
       {
                        if(FD_ISSET(sock,&fdread)) {
                           sresult = 2*PACKET_SIZE;
                           if (rtp_recvfrom(&rcvr,(void
*)&msg,&sresult) > -1) {
                                   printf("\talgorithms:
%s\n",msg.data);

select_scrambling_algo(&clientalgo,&serveralgo,msg.data,inpu
t_key);

message_init(&msg,MSG_TYPE_ALGO,0);
                                   msg.msg_hdr.seq_num =
clientalgo;
                                   rtp_sendto(&snd,(void
*)&msg,sizeof(message_t));
                                   printf("sending client algo:
%d server algo: %d\n",clientalgo,serveralgo);
                           }
                        }
        } else {
          /* timedout, so send using default algorithm */
             printf("\ttimed out\n");
             exit(0);
        }
        //send selected algorithm info
     }

    /* read words from dictionary, then send them off */
    ctr = 0;
    ts.tv_sec = 0;
    ts.tv_nsec = USEC_RATE;
    while ((len = fread(word,1,PACKET_SIZE,dict)) > 0) {
       int i = 0;
       //scramble the word data here
       message_init(&msg,MSG_TYPE_DATA,len);
       memcpy((void *)msg.data,(void *)word,len);
       for(i = 0; i < PACKET_SIZE/(2*sizeof(long)); i++) {
          server_code(serveralgo,(long
*)&(word[i*2*sizeof(long)]),(long *)tea_key,(long
*)&(msg.data[i*2*sizeof(long)]));
       }
       msg.data[len] = '\0';
```

```
        msg.msg_hdr.seq_num = ctr;
        printf("\nsending scrambled packet %d of length
%d\n",ctr,len);
        if(ctr  < 10 ) {
          server_print_hex(msg.data,len);
          printf("\noriginal\n");
          server_print_hex(word,len);
          printf("\n");
        }
        rtp_sendto(&snd,(void *)&msg,sizeof(message_t));
        ctr++;
        if(len < PACKET_SIZE)
           break;
        if(ctr % 50 == 0)
          nanosleep(&ts,NULL);
    }

  if (ADDR_IS_MULTICAST(rcvr_addr.s_addr)) {
    leave_group(sock, mreq, argv[0]);
  }
  close(sock);
  return 0;
}

int
do_permissions(const char *username, const char *filename) {

  char *permissionsfile = PERMISSIONS_FILE;
  FILE *fp = NULL;
  fpos_t curpos;
  char data[MAXFILESTR+1];
  char givenuser[MAXFILESTR+1];
  char givenfile[MAXFILESTR+1];
  char givenpermissions[MAXFILESTR+1];
  int counter = -1;

  if(initscanner(permissionsfile,&fp) != 0)
     return 1; /* permissions file does not exist, so allow
everyone */
  deinitscanner(&fp);

  if(initwriter(permissionsfile,&fp) != 0)
     return 1; /* permissions file does not exist, so allow
everyone */

  while(getwriterdata(&fp,&curpos,data,MAXFILESTR) == 0) {
    my_awkstr(data," ",1,givenfile,MAXFILESTR);
    if(strlen(givenfile) == strlen(filename) &&
strcmp(givenfile,filename) == 0) {
        my_awkstr(data," ",2,givenuser,MAXFILESTR);
        if(strlen(givenuser) == strlen(username) &&
strcmp(givenuser,username) == 0) {
```

```
         /* found a match for username and filename */
         my_awkstr(data," ",3,givenpermissions,MAXFILESTR);
         if(strstr(givenpermissions,"inf") != NULL) {
            deinitwriter(&fp);
            return 1;
         }
         counter = atoi(givenpermissions);
         if(counter > 0) {
           snprintf(data,MAXFILESTR,"%s   %s
%d",givenfile,givenuser,counter-1);
           replace_in_file(fp,&curpos,data);
           deinitwriter(&fp);
           return 1;
         }
         deinitwriter(&fp);
         return 0;

       } /* end of found match */

    }

  }
  deinitwriter(&fp);
  return 0;
}

void
usage(char *string) {

  printf("usage: %s dest_ip dest_port filename username
samplingrate\n",
      string);
  exit(1);

}


void
leave_group(int sock, struct ip_mreq mreq, char *name) {
  int ret;

  ret = setsockopt(sock, IPPROTO_IP, IP_DROP_MEMBERSHIP,
&mreq, sizeof(mreq));
  if (ret < 0) {
     fprintf(stderr, "%s: Failed to leave multicast group",
name);
     perror("");
  }
}

/* read the key from KEYS_FILE */
int
```

```
get_client_key(const char *username,unsigned char
**clientkey){

  char *keysfile = KEYS_FILE;
  FILE *fp = NULL;
  char data[MAXFILESTR+1];
  char givenuser[MAXFILESTR+1];
  char key[MAXFILESTR+1];

  *clientkey = NULL;
  if(initscanner(keysfile,&fp) != 0)
     return -1; /* cannot open keys file */

  while(getscannerdata(&fp,data,MAXFILESTR) == 0) {
    my_awkstr(data," ",1,givenuser,MAXFILESTR);
    if(strlen(givenuser) == strlen(username) &&
strcmp(givenuser,username) == 0) {
         /* found a match for username*/
         my_awkstr(data," ",2,key,MAXFILESTR);
         right_left_trim(key);
         *clientkey = (unsigned char *) strdup(key);
         break;
    }
  }
  deinitscanner(&fp);
  return 0;
}

/* generate session key */
int
get_session_key(unsigned char **sessionkey,unsigned char
*teakey){

  char tmpfile[TMP_FILENAME_LEN+1];
  FILE *fp = NULL;
  char data[MAXFILESTR+1];
  char sdata[MAXFILESTR+1];

  //create_tmp_file(tmpfile,TMP_FILENAME_LEN);
  snprintf(tmpfile,TMP_FILENAME_LEN,"/var/sessionkey");
  erase_file(tmpfile);
  my_cmd("(cat /dev/random | od --read-bytes=32 --width=32 -
x | awk '{ print $3$4$5$6$7$8$9$10$11$12$13$14$15$16$17\"
\"$5$9$13$15 }') 2> /dev/null > %s ",tmpfile);

  if(*sessionkey != NULL) {
    free(*sessionkey);
    *sessionkey = NULL;
  }
  if(initscanner(tmpfile,&fp) != 0)
     return -1; /* cannot open keys file */
```

```
  while(getscannerdata(&fp,data,MAXFILESTR) == 0) {
         right_left_trim(data);
         my_awkstr(data," ",1,sdata,MAXFILESTR);
         *sessionkey = (unsigned char *)strdup(sdata);
         my_awkstr(data," ",2,sdata,MAXFILESTR);
         memcpy((void *)teakey,(void *)sdata,16);
         teakey[16]='\0';
         break;
  }
  deinitscanner(&fp);
  //remove_file(tmpfile);
  return 0;
}

int
select_scrambling_algo(int *clientalgo,int *serveralgo,const
char *supportedalgos,unsigned char *sessionkey){
  FILE *fp = NULL;
  int totalsupported = 0,totalunbroken=0;
  char *tmp = NULL;
  char *tmp1 = (char *)supportedalgos;
   char buffer[65];
   char output[65];
   char broken[MAXFILESTR+1];
  int i;
  unsigned int seed;
  char salgos[MAXFILESTR+1];

  snprintf(salgos,MAXFILESTR,"%s",supportedalgos);
  fp = fopen(BROKEN_FILE,"r");
  while(fgets(broken,MAXFILESTR,fp) != NULL) {
      if(strlen(broken) > 0)
         break;
  }
  fclose(fp);

  while((tmp = strstr(tmp1,"\n")) != NULL) {
    totalsupported++;
    tmp1 = tmp+1;
  }
  if(tmp1 != NULL && *tmp1 != '\0')
    totalsupported++;
  totalunbroken = totalsupported;
  sscanf(sessionkey,"%x",&seed);
  srand(seed);
  printf("Total algorithms supported by client
%d\n",totalsupported);
  *clientalgo=1+(int)
(totalsupported*rand()/(RAND_MAX+1.0));

  printf("Random client algorithm %d\n",*clientalgo);
```

```
  tmp1 = (char *)salgos;
  tmp = NULL;
  for(i =0; i < *clientalgo && tmp1 != NULL ; i++) {
    if(tmp != NULL)
       tmp1 = tmp+1;
    tmp = strstr(tmp1,"\n");
  }
  *serveralgo = 0;
  if(tmp != NULL)
     *tmp = '\0';
  printf("Selected secret %s\n",tmp1);
  for(i = 0; i < MAX_SUPPORTEDALGOS;i++) {
      snprintf(buffer,64," %d ",i+1);
      snprintf(output,64,"%s",crypt(buffer,SERVERKEY));
      printf("secret %d %s\n",i+1,output);
      if(strcmp(output,tmp1) == 0) {
          *serveralgo = i+1;
          if(strstr(broken,buffer) != NULL) {
              printf("Algorithm %s is broken\n",buffer);
              totalunbroken--;
              if(totalunbroken == 0) {
                 printf("Client algorithms are hacked\n");
                 printf("Aborting this transmission\n");
                 exit(0);
              }
              *clientalgo = ( *clientalgo + 1 <
totalsupported ? *clientalgo+1 : *clientalgo - 1);
              if( *clientalgo < 1) {
                 //this should not be happening
                 printf("Client algorithms are hacked\n");
                 printf("Aborting this transmission\n");
                 exit(0);
              }

snprintf(salgos,MAXFILESTR,"%s",supportedalgos);
              tmp1 = (char *)salgos;
              tmp = NULL;
              for(i =0; i < *clientalgo && tmp1 != NULL ;
i++) {
                   if(tmp != NULL)
                       tmp1 = tmp+1;
                   tmp = strstr(tmp1,"\n");
             }
             *serveralgo = 0;
             if(tmp != NULL)
               *tmp = '\0';
             continue;
          }
          break;
      }
  }
  if( *serveralgo == 0) {
```

```
                printf("Matching algorithm not found\n");
                printf("Aborting this transmission\n");
                exit(0);
  }
  return 0;
}

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

/************************************************************
********************/
/**********Tea Routine -
Original**********************************************/

/**********Decode
Routine*****************************************************
**/

void server_decode(int type,long* in,long* k,long *out)  {
 unsigned long n=32, sum, y=in[0], z=in[1], delta=0x9e3779b9
;

 switch (type) {

 case 0:
        sum=delta<<5 ;
                       /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                       /* end cycle */
        out[0]=y ; out[1]=z ;
          break;

 case 1:
        delta=0xae3778b9 ;
        sum=delta<<5 ;
                       /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                       /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 case 2:
        delta=0xae3778b9 ;
        sum=delta<<5 ;
```

```
                      /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ^
(0xabcd0123) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ^
(0xabcd0123) ;
        sum-=delta ;  }
                      /* end cycle */
        out[0]=y ; out[1]=z ;
        break;
 case 3:
         y=in[1];
         z=in[0];
        delta=0x9e3779b9 ;
        sum=delta<<5 ;
                      /* start cycle */
        while (n-->0) {
           z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]);
           y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]);
           sum-=delta ;  }
                      /* end cycle */
        out[0]=y ; out[1]=z ;
         break;
 case 4:
         y=(in[1]) ^ (0x8abc);
         z=(in[0]) ^ (0x7def);
        delta=0x9e3779b9 ;
        sum=delta<<5 ;
                      /* start cycle */
        while (n-->0) {
          z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]);
          y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]);
          sum-=delta ;  }
                      /* end cycle */
        out[0]=y ; out[1]=z ;
         break;
 case 5:
         y=(in[1])^(0x8abc);
         z=(in[0])^(0x7def);
        delta=0x9e377dab ;
        sum=delta<<5 ;
                      /* start cycle */
        while (n-->0) {
          z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]);
          y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]);
          sum-=delta ;  }
                      /* end cycle */
        out[0]=y ; out[1]=z ;
         break;
 case 6:
         y=(in[1])^(0x8abc);
         z=(in[0])^(0x7def);
```

```
        delta=0x9e377aab ;
        sum=delta<<5 ;
                       /* start cycle */
        while (n-->0) {
          z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]);
          y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]);
          sum-=delta ;  }
                       /* end cycle */
        out[0]=y ; out[1]=z ;
         break;
 case 7:
        delta=0xdab778b9 ;
        sum=delta<<5 ;
                       /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                       /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 case 8:
        delta=0xaab778b9 ;
        sum=delta<<5 ;
                       /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                       /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 case 9:
        delta=0xaabdabb9 ;
        sum=delta<<5 ;
                       /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                       /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 case 10:
        delta=0xdabaabb9 ;
        sum=delta<<5 ;
                       /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                       /* end cycle */
```

```
        out[0]=y ; out[1]=z ;
          break;
 case 11:
        delta=0xdabaabab ;
        sum=delta<<5 ;
                        /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                        /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 case 12:
        delta=0xdabaabdb ;
        sum=delta<<5 ;
                        /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                        /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 case 13:
        delta=0xefbaabdb ;
        sum=delta<<5 ;
                        /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                        /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 case 14:
        delta=0xabcdefab ;
        sum=delta<<5 ;
                        /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                        /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 case 15:
        delta=0xfedcabfe ;
        sum=delta<<5 ;
                        /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
```

```
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                       /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 case 16:
        delta=0x1a2b3c4d ;
        sum=delta<<5 ;
                       /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                       /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 default:
         break;
 }

}



/******************Encode
Routine***********************************************/
/*Routine, written in the C language, for encoding with key
k[0] - k[3].
Data in v[0] and v[1]. */


void server_code(int type,long* in, long* k,long *out)  {
unsigned long y=in[0],z=in[1], sum=0,   /* set up */
 delta=0x9e3779b9, n=32 ;             /* a key schedule
constant */

switch(type) {

case 0:
     while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
             }
    out[0]=y ; out[1]=z ;
    break;
case 1:
     delta=0xae3778b9;             /* a key schedule
constant */
```

```
    while (n-->0) {                        /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;
case 2:
     delta=0xae3778b9;             /* a key schedule
constant */
    while (n-->0) {                        /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ^
(0xabcd0123) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ^
(0xabcd0123) ;   /* end cycle */
              }
    out[0]=y ; out[1]=z ;
     break;
case 3:
    delta=0x9e3779b9;             /* a key schedule constant
*/
    while (n-->0) {                        /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1])  ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3])  ;   /* end
cycle */
              }
    out[0]=z ; out[1]=y ;
     break;
case 4:
    delta=0x9e3779b9;             /* a key schedule constant
*/
    while (n-->0) {                        /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1])  ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3])  ;   /* end
cycle */
              }
    out[0]=z^0x7def ; out[1]=y^0x8abc ;
     break;
case 5:
    delta=0x9e377dab;             /* a key schedule constant
*/
    while (n-->0) {                        /* basic cycle
start */
```

```
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1])  ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3])  ;   /* end
cycle */
              }
    out[0]=z^0x7def ; out[1]=y^0x8abc ;
     break;
case 6:
    delta=0x9e377aab;             /* a key schedule constant
*/
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1])  ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3])  ;   /* end
cycle */
              }
    out[0]=z^0x7def ; out[1]=y^0x8abc ;
     break;
case 7:
     delta=0xdab778b9;             /* a key schedule
constant */
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;

case 8:
     delta=0xaab778b9;             /* a key schedule
constant */
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;

case 9:
     delta=0xaabdabb9;             /* a key schedule
constant */
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
```

```
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;

case 10:
     delta=0xdabaabb9;             /* a key schedule
constant */
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;

case 11:
     delta=0xdabaabab;             /* a key schedule
constant */
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;

case 12:
     delta=0xdabaabdb;             /* a key schedule
constant */
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;

case 13:
     delta=0xefbaabdb;             /* a key schedule
constant */
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
```

```
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;

case 14:
     delta=0xabcdefab;             /* a key schedule
constant */
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;

case 15:
     delta=0xfedcabfe;             /* a key schedule
constant */
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;

case 16:
     delta=0x1a2b3c4d;             /* a key schedule
constant */
    while (n-->0) {                       /* basic cycle
start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;   /* end
cycle */
     }
    out[0]=y ; out[1]=z ;
     break;

default:
     break;
}

}
```

```
void server_print_hex(const unsigned char *str,int len) {
 int i =0;
     for(i=0 ; i < len ; i++,str++)
          printf("%.2x",*str);

}
```

Appendix F: Secure driver

```
/*
 Modified by Deepali Holankar to implement de-scrambling
parameters intialization and de-scrambling algorithms
 *   Intel i810 */

#include "secure_driver.h"


/* "software" or virtual channel, an instance of opened
/dev/dsp */
struct i810_state {
        struct security_info m_secure;
};


/* in this loop, dmabuf.count signifies the amount of data
that is waiting to be dma to
   the soundcard.  it is drained by the dma machine and
filled by this loop. */
static ssize_t i810_write(struct file *file, const char
*buffer, size_t count, loff_t *ppos)
{
     struct dmabuf *dmabuf = &state->dmabuf;

        /*decode here before copying to dma buffer */
     if (copy_from_user(secure->data,buffer,count)) {
               if (!ret) ret = -EFAULT;
               return ret;


        }
        for(x = 0;  x < count ; x = x + PACKET_SIZE) {
             for(cnt= 0; cnt < PACKET_SIZE/(2
*sizeof(long));cnt++) {
                 receiver_decode(secure->algo,(long
*)&(secure->data[x+(2*cnt*sizeof(long))]),(long *)secure-
>teakey,

(long *)&(secure->data[x+(2*cnt*sizeof(long))]));
             }
        }
        buffer = (const char *)secure->data;

        memcpy(dmabuf->rawbuf+swptr,buffer,cnt);

static int i810_ioctl(struct inode *inode, struct file
*file, unsigned int cmd, unsigned long arg)
{
     struct security_info *secure = &state->m_secure;

case SNDCTL_DSP_SECURITY:
```

```
#ifdef DEBUG
          printk("SNDCTL_DSP_SECURITY\n");
#endif
          memcpy((void *)secure, (void
*)arg,sizeof(security_info));
                return 0;
}

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

/* server 5 -> client 1 , server 6 -> client 2,




   server 7-> client 3, server 9 -> client 4 server 10 ->
client 5 */
/*******************************************************
*******************/
/**********Tea Routine -
Original*********************************************/

/**********Decode
Routine*************************************************
**/

void receiver_decode(int type,long* in,long* k,long *out)  {
 unsigned long n=32, sum, y=in[0], z=in[1], delta=0x9e3779b9
;

 switch (type) {

 case 1:
         y=in[1]^0x8abc;
         z=in[0]^0x7def;
        delta=0x9e377dab ;
        sum=delta<<5 ;
                      /* start cycle */
        while (n-->0) {
          z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]);
          y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]);
          sum-=delta ;  }
                      /* end cycle */
        out[0]=y ; out[1]=z ;
         break;
 case 2:
            y=(in[1])^(0x8abc);
            z=(in[0])^(0x7def);
        delta=0x9e377aab ;
        sum=delta<<5 ;
                      /* start cycle */
```

```
        while (n-->0) {
          z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]);
          y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]);
          sum-=delta ;  }
                        /* end cycle */
        out[0]=y ; out[1]=z ;
            break;
 case 3:
        delta=0xdab778b9 ;
        sum=delta<<5 ;
                        /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }


                        /* end cycle */
        out[0]=y ; out[1]=z ;
                break;
 case 4:
        delta=0xaabdabb9 ;
        sum=delta<<5 ;
                        /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                        /* end cycle */
        out[0]=y ; out[1]=z ;
          break;
 case 5:
        delta=0xdabaabb9 ;
        sum=delta<<5 ;
                        /* start cycle */
        while (n-->0) {
        z-= ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        y-= ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
        sum-=delta ;  }
                        /* end cycle */
        out[0]=y ; out[1]=z ;
                break;
 default:
         break;
 }

}
```