TABLE OF CONTENTS

1 LIST OF SOFTWARE

1.1 FFMPEG

1.1.1 Download and Installation

Download ffmpeg from CVS.

```
$ cvs -z9 -d:pserver:anonymous@mplayerhq.hu:/cvsroot/ffmpeg co ffmpeg
```

This folder includes source codes and makefiles for building and installing ffmpeg.

To install ffmpeg in a personal directory, rather then the default of `/usr/bin`

```
$ ./configure --prefix=/home/cguo/ffmpeg_install
$ make
$ make install
```

1.1.2 Binaries and Library Files

After the built, the execution file will be stored in the `/ffmpeg` directory. The library files (.a files) and header files (.h files) are in `/ffmpeg/libavcodec`, `/ffmpeg/libavformat`, `/ffmpeg/libutil` directories respectively. In order to build a custom program using ffmpeg functions and codec, the library files and their path need to be referenced in the makefile.

Note: the header files in the `/ffmpeg_install/include` directory are not complete.

1.1.3 Usage

To convert a video file to MPEG4 format, the following command can be used.

```
$ ./ffmpeg -i input_file output_file

Usefule options:
      -av   disable audio
      -b    bit rate
      -f    frame rate
      -g    gop size
      -s    output image size
```

1.2 XML READER AND PARSER

1.2.1 Download and Installation

Download zip or tar file from ftp://xmlsoft.org/libxml2.

To install the binaries to a personal directory, instead of the default of /usr/bin, unpack the files to a new directory

```
$ ./configure --prefix /home3/cguo/myxml/xmlinst
$ make
$ make install
```

The files will be installed in
`/home3/cguo/myxml/xmlinst/lib`
`/home3/cguo/myxml/xmlinst/bin`
`/home3/cguo/myxml/xmlinst/include`

1.3 ENCRYPTION

Encryption code can be downloaded from http://www.cypherspace.org/adam/rsa/rc4.c. The encryption algorithm is symmetric; therefore, same algorithm is used for decryption.

2 IMPLEMENTATIONS

This section outlines the implementations of the project from the individual components (section number) to the final program (section number). The main problems are defined in section 2.1 and each component targets a specific part of the problems defined. The final program includes encryption (`Encrypt_xml.c`) and decryption (`Decrypt_xml.c`) of packet data and frame rate conversion example (`RateConvert_xml.c`).

2.1 OBJECTIVES

- The MPEG4 video file should be separated into packets.
- The packets have to be encrypted to avoid security threats in the transcoder.
- Each packet should have an xml header that contains all the parameters needed by the transcoder to perform streaming operations.
- The end user should be able to reconstruct a playable MPEG4 file from the encrypted and/or modified packet data.

2.2 INDIVIDUAL COMPONENTS

In this section, the problems are divided into the following areas: read/write MPEG4 packets (2.2.1), encrypt/decrypt packet data (2.2.2), parse/generate xml file (2.2.3). In addition, the xml header structure for this specific project is presented in 2.2.4, and the encrypted MPEG4 file structure is discussed in 2.2.5.

2.2.1 Read and write MEPG4 packets

Packet information are extracted from the MPEG4 files using ffmpeg `av_read_frame()` function. One problem with this function is that the display time stamp (dts) and the presentation time stamp (pts) are not extracted correctly. It is not known that whether this is an issue with ffmpeg or that dts and pts are simply not available within the packet. This problem will not affect frame rate conversion calculations.

open input file	`av_open_input_file()`
update stream information	`av_file_stream_info()`
read packet	`av_read_frame()`

Table 1: FFMPEG packet extracting functions

To generate a playable MPEG4 file, `av_write_frame()`is used. The table below outlines the process and ffmpeg functions used to output a video file. The `guess_format()` function extracts the output file format from its extension. Although the output file for this application is always MPEG4, this function is called to set up the necessary parameters such as `oformat` when generating output files.

detect output format	`guess_format()`
allocate media context	`av_alloc_format_context()`
initialize stream and codec	`av_new_stream()`
write header	`av_write_header()`
write frame	`av_write_frame()`
write trailer	`av_write_trailer()`
clean up pointers	`av_free()`

Table 2: FFMPEG packet writing functions

2.2.2 Encrypt and Decrypt Packet Data

The packet data of each frame is encrypted using rc4 algorithm introduced in List of Software.

```
// obtain parameters
encryption_key = user defined;

// encryption initialization
encrypt_init(encryption_key);

while (read new packet) {
      encrypt packet;
}
```

Table 3: Encryption/Decryption algorithm

2.2.3 Read and Write XML

It is easier to generate an error-free xml file from an existing dummy xml file. The dummy xml file has to have a complete structure. The dummy file used in this project is the same as the one in Table 5.

The main functions used to read and write an xml header file is summarized in Table 4.

read and validate xml file	xmlCtxtReadFile()
get xml nodes	xmlDocGetRootElement()
set/read xml node content	xmlNodeSetContent()/xmlNodeGetContent()
save xml file	xmlSaveFile()
close file	xmlFreeDoc()

Table 4: Functions that are used to read/write XML files

2.2.4 XML Header Structure

The MPEG4 packet header is stored using xml. Currently, the elements in the xml are the same as the packet structure, `AVPacket`, in ffmpeg. Other parameters can be added easily to xml if they are needed by the transcoder.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE packetStream SYSTEM "pckexample.dtd">
<packetStream>
<packet>
    <pts>299</pts>
    <dts>299</dts>
    <size>296</size>
    <stream_index>0</stream_index>
    <duration>1</duration>
    <pos>570929</pos>
    <flags>0</flags>
    <data/>
</packet>
</packetStream>
```

Table 5: XML header file

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT packetStream (packet)>
<!ELEMENT packet (pts, dts, size, stream_index, duration, pos, flags,
data)>
<!ELEMENT pts (#PCDATA)>
<!ELEMENT dts (#PCDATA)>
<!ELEMENT size (#PCDATA)>
<!ELEMENT stream_index (#PCDATA)>
<!ELEMENT duration (#PCDATA)>
<!ELEMENT pos (#PCDATA)>
<!ELEMENT flags (#PCDATA)>
<!ELEMENT data (#PCDATA)>
```

Table 6: XML header DTD (Document Type Definitions)

2.2.5 Encrypted XML-MPEG4 file structure

The encrypted MPEG4 file with xml headers has the following structure. It starts with a binary header which contains the size (width and height), frame rate, bit rate and codec id. All of the fields are needed to reconstruct a playable MPEG4 file after decryption. Each packet begins with a 32-bit number indicating the size of the xml header file, followed by xml header file in text format, and then the packet data in binary format.

file header	20 bytes
xml header size for packet 1	4 bytes
xml header for packet 1	300 ~ 310 bytes
packet 1 data	various
xml header size for packet 2	4 bytes
xml header for packet 2	300 ~ 310 bytes
packet 2 data	various
…	…

Table 7: Encrypted XML-MPEG4 file structure

2.3 STRUCTURE OF THE PROGRAMS

2.3.1 Encryption and Decryption

The encryption and decryption program can be called from the command line as follows.

```
$ ./xmlencrypt input_file encryption_key output_file
$ ./xmldecrypt input_file encryption_key output_file
```

The encryption and decryption program takes three input parameters, namely input file name, encryption key, and output file name. Note that the input file to the encryption program and output file from the decryption program should have the same format, (i.e. .mp4). The program might not work correctly if different formats are chosen. The maximum length of the encryption key is 256 char or 256 bytes.

The base for the encryption program (`Encrypt_xml.c`) is `pktdumper.c` in the `/ffmpeg` directory, whereas the base for the decryption program (`Decrypt_xml.c`) is `output_example.c` in the same directory. Additional definitions and functions, including encryption and xml parser, are in the `output_xml.h` file.

Encryption and Decryption steps are summarized in Table 8 and 9 respectively.

```
Step 1: Write file header to the output file
Step 2: Read MPEG4 packet (2.2.1)
Step 3: Encrypt packet (2.2.2)
Step 4: Parser packet info to xml header (2.2.3)
Step 5: Write xml header size to the output file
Step 6: Write xml header to the output file
Step 7: Write encrypted packet to the output file
Step 8: Repeat steps 2~7 while there is more packet
```

Table 8: Steps for generating an encrypted MPEG4 file with XML packet headers

```
Step 1: Read file header from the input file
Step 2: Initialize output file (2.2.1)
Step 3: Read xml header size
Step 4: Read and parse xml header (2.2.3)
Step 5: Read and decrypt packet data (2.2.2)
Step 6: Write MEPG4 packet/frame (2.2.1)
Step 7: Repeat steps 3~6 till the end of the input file
```

Table 9: Steps for generating a playable MPEG4 file
from the encrypted XML-MPEG4 file

2.3.2 Frame rate conversion

One of the easiest transcoding operations is frame rate conversion. This section describes the frame conversion program (`RateConvert_xml.c`), which takes an encrypted XML-MPEG4 file as the input.

The rate conversion program can be called from the command line as follows.

```
$ ./xmlrateconvert input_file options new_rate output_file

Options
      -f    indicates new_rate is the new frame rate (fps)
      -p    indicates new_rate is a percentage (0~100) of the original
            frame rate
```

The rate conversion steps are summarized in the following table.

```
Step 1: Read file header from the input file
Step 2: Obtain user input parameters
Step 3: Do frame conversion calculations (2.3.2)
Step 4: Write file header to the output file
Step 5: Read xml header size
Step 6: Read and parse xml header (2.2.3)
Step 7: Run rate conversion algorithm (2.3.2)
If “drop frame”,
      Step 8: Repeat steps 5~7 till the end of the input file
If “do not drop”,
      Step 8: Write xml header size to the output file
      Step 9: Write xml header to the output file
      Step 10: Write encrypted packet to the output file
      Step 11: Repeat steps 5~7 till the end of the input file
```

Table 10: Steps for performing rate conversion operation
on the encrypted XML-MPEG4 file

Since one packet corresponds to one frame of the original video file, frame conversion is achieved by dropping certain packets from the file. However, frames cannot be deleted at random. I frames and some P frames should never be dropped since all the subsequent

frames rely on them to decode. Only P frames that come before an I frame can be ignored without noticeable degradation of the video file.

For this reason, the program has to know the size of GOP (group of picture) in order to decide which frames to drop. GOP is not stored within the packets but it is hardcoded in the program. To change this hard coded number, go to `RateConvert_xml.c`, line 64.

```
// obtain parameters
frame_rate_old = from original file;
frame_rate_new = user defined;
gop = 5;

// calculate number of P frames to drop per group
no_reduced_P_per_group = gop – frame_rate_new x gop/frame_rate_old;
no_reduced_P_base = floor(no_reduced_P_per_group);
no_reduced_P_q = no_reduced_P_per_group – no_reduced_P_base;

FLOOR1 = 1;
FLOOR2 = 0;
while (read new packet) {
      if ((frame_index - (FLOOR*gop-no_reduced_P_base))==0)  {
          if (I frames) {
              do not drop;
          }
          if (no_reduced_P_base == 0) {
              no_reduced_P_base = floor(no_reduced_P_per_group);
              FLOOR ++;

              if (round(no_reduced_P_q*FLOOR) - FLOOR1 == 1) {
                  no_reduced_P_base ++;
                  FLOOR1 ++;
              }
          }
          else {
              no_reduced_P_base --;
              drop frame;
          }
      }
      else {
          do not drop;
      }
}
```

Table 11: Rate conversion algorithm

3 PROBLEMS AND POSSIBLE INPROVEMENTS

The lower frame rate limit is 17 for ffmpeg output files. Lowering the frame rate more will introduce serve distortions to the output file. The reason for this is not known at this point. The rate conversion program does not limit the output frame rate with the hope that this problem can be solved later.

When using rate conversion or encryption programs with an encrypted MPEG4 file, there will be error messages saying "header damage". This is because ffmpeg tries to decode the first couple of packets to extract the size of the frame. If the MPEG4 file is encrypted, decoding will not work properly; hence the error output. The error message can be ignored, but the size information is needed for both rate conversion and encryption. The short term solution is to save the size of the output frame to a text file and read it when ffmpeg fail to retrieve it. A better way is to hide the size information somewhere in the packet, but, up to this point, I have not found a good place for it.

Cindy's Note: After adding the appropriate file header and XML header, the above problem disappeared since the program no longer calls ffmpeg functions to extract size information.

As mentioned earlier, GOP is not stored within the packet. The algorithm uses a default value GOP = 5 to do all the rate conversion calculations. It is possible to change the default value by adding a user input parameter; however, one cannot expect the user to know the correct GOP of the input file. GOP can be extracted since the packet has a flag to indicate frame type. Though inefficient, it is the best solution I see so far.

Each XML header is about 300 bytes, which could make the encrypted output a lot bigger than the original file. It is noticed, however, that not all of the elements defined are necessary. <pts> and <dts> are not correct and are not used; the real data is not in <data>; and it is not clear what <pos> is used for (and it is always -1 or 0, i.e. undefined). Removing the unused tags will reduce the size of the file by about 100 bytes.

4 USEFUL WEB LINKS

ffmpeg documentation: http://ffmpeg.sourceforge.net/ffmpeg-doc.html
describes how to use ./ffmpeg and contains the supported codec

ffmpeg doxygen: http://www.irisa.fr/texmex/people/dufouil/ffmpegdoxy/
useful in tracing convoluted ffmpeg codes

xml parser documentation: http://xmlsoft.org/docs.html
contains function definition of libxml2 and code examples

APPENDIX A: PACKAGE DESCRIPTION

Files included:

RateConvert.c	converts the frame rate of the a mp4 file without xml headers
Encrypt.c	encrypts/decrypts mp4 packets without xml headers
output_cindy.h	header files for RateConvert.c and Encrypt.c
RateConvert_xml.c	converts the frame rate of a mp4 file with xml headers
Encrypt_xml.c	encrypts a playable mp4 file and adds xml headers
Decrypt_xml.c	generates a playable mp4 file from the encrypted mp4 file with xml headers
output_xml.h	header file for the *_xml.c files
Makefile	used to compile the source files
pckexample.xml	sample xml file, needed for header extraction
pckexample.dtd	DTD for the xml file, needed for validation
work.doc	documentation

Notes:

Makefile:	make sure to check the SRC directory and xml libary directory before using it
output_cindy.h	contains xml functions although they are not used by the .c files

Usage:

make rateconvert	makes RateConvert.c
make pktencrypt	makes Encrypt.c
make xmlrateconvert	makes RateConvert_xml.c
make xmlencrypt	makes Encrypt_xml.c
make xmldecrypt	makes Decrypt_xml.c

./rateconvert input_file -f/-p new_rate
./pktencrypt input_file encryption_key

./xmlrateconvert input_file -f/-p new_rate output_file
./xmlencrypt input_file encryption_key output_file
./xmldecrypt input_file encryption_key output_file