Efficient Frequency Domain Selective Scrambling of Digital Video

P- 2168

A PROJECT REPORT

Submitted by

SWAATHIGA .K.N	**71204205055**
GOMATHI.M	**71204205008**

In partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2008

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"Efficient Frequency Domain Selective Scrambling of Digital Video"** is the bonafide work of "K.N.Swaathiga (71204205055) and M.Gomathi (71204205008)" who carried out the project work under my supervision.

(Signature)	(Signature)
Dr. S.Thangasamy	Mrs.S.Vani, Lecturer.
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Dept of Information Technology,	Dept of Information Technology,
Kumaraguru College of Technology,	Kumaraguru College of Technology,
Coimbatore- 641 006.	Coimbatore - 641 006.

Submitted for viva-voice examination held on 24.04.08

INTERNAL EXAMINER EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our sincere thanks to our chairman **Padmabhushan Arutselvar Dr. N. Mahalingam B.Sc, F.I.E** and vice chairman **Prof. K. Arumugam B.E., M.S., M.I.E.,** for all their support and ray of strengthening hope extended. We are immensely grateful to our principal **Dr. Joseph V. Thanikal M.E., Ph.D., PDF., CEPIT.,** for his invaluable support to the outcome of this project.

We are deeply obliged to our Head of the Department **Dr. S.Thangasamy,** for his concern and implication during the project course.

We extend our thanks to our project coordinator **Prof.K.R.Baskaran,** Asst Professor, Department of Information Technology and our guide **Mrs.S.Vani,** Lecturer, Department of Information Technology for their support given to us through this project.

We thank the teaching and non-teaching staffs of our Department for providing us the technical support in the duration of our project.

We express our humble gratitude and thanks to our beloved parents and family members who have supported and helped us to complete the project and our friends, for lending us valuable tips, support and co-operation throughout the project work.

ACKNOWLEDGEMENT

DECLARATION

DECLARATION

We,

SWAATHIGA.K.N	**71204205055**
GOMATHI.M	**71204205008**

hereby declare that the project entitled **" Efficient Frequency Domain Selective Scrambling of Digital Video",** submitted in partial fulfillment to Anna University as the project work of Bachelor of Technology (Information Technology) Degree, is a record of original work done by us under the supervision and guidance of Department of Information Technology, Kumaraguru college of Technology, Coimbatore.

Place: Coimbatore

Date:

(Swaathiga.K.N)

(Gomathi.M)

TABLE OF CONTENTS

TABLE OF CONTENTS

ABSTRACT

This project entitled 'Efficient Frequency Domain Selective Scrambling of Digital Video' presents joint encryption and compression framework in which video data are scrambled efficiently in the frequency domain by employing selective bit scrambling, block shuffling and block rotation of the transform coefficients and motion vectors.

This project consists of four modules. The modules are Selective Bit Scrambling, Block Shuffling, Block Rotation and Security Analysis. All the four modules operate on the images that are constructed from the video.

This new approach is very simple to implement, yet provides considerable levels of security and has no adverse impact on the compression efficiency. Furthermore, it allows transcodability /scalability and some other content processing functionality without having to access the cryptographic key and to perform decryption and re-encryption.

The Java Standard Development Kit along with Java Media Framework is used for the development of the project. The java Media Framework enables us to access and evaluate various multimedia formats efficiently.

LIST OF FIGURES

LIST OF ABBREVIATIONS

JPEG	-	Joint Photographic Experts Group
BMP	-	Bit Map Image
IPR	-	Intelluctual Property Right
DVD	-	Digital Versatile Disk
MPEG	-	Motion Pictures Experts Group
DES	-	Digital Encryption Standard
PDA	-	Personal Digital Assistant
2D	-	Two Dimensional
3D	-	Three Dimensional
GB	-	Giga Byte
MB	-	Mega Byte
CD	-	Compact Disk
FDD	-	Floppy Disk Drive
ROM	-	Read Only Memory
J2SDK	-	Java Software Development Kit
AVI	-	Audio Video Interleaved

CHAPTER 1
INTRODUCTION

1.1 GENERAL

It is recognized that digital image encryption presents a set of issues, aside from security, that are unique in the data cryptography field. A digital image-scrambling scheme should have a relatively simple implementation, amenable to low-cost decoding equipment and low-delay operation for real-time interactive applications. It should have a minimum adverse impact on the compressibility of the image.

It should preferably be independent of the bitstream compression selected for the image, and allow compression transcodability/scalability without having to decrypt. It should provide good overall security, although it may also be preferable in some systems to allow non-authorized users a level of transparency, both to entice them to pay for full transparency, and to discourage code-breaking.

Our proposed digital video scrambling system aims to meet the objectives outlined above. Fig. 1.1 shows a general architecture of our proposed scrambling system. At the encoder, the input video signal is first transformed into the frequency domain, by performing, e.g., 2D/3D wavelet transform or Discrete Cosine Transform (DCT). The input signal may be original video frame or motion compensated residual signal.

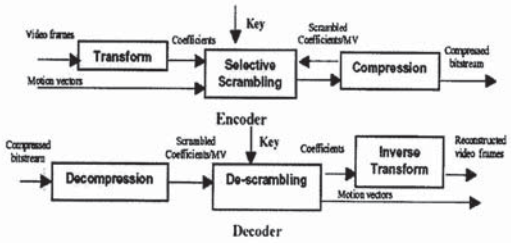


FIG 1.1 GENERAL ARCHITECTURE OF THE PROPOSED SCRAMBLING SYSTEM

The transform coefficients are then divided into blocks/segments and are subject to selective scrambling, which may consist of all or some of the following operations: selective bit (e.g., sign bits) scrambling, random sign change, block shuffling, block rotation, coefficient shuffling within a subband segment, etc. The motion vectors, if any, are also subject to random sign change and shuffling. A cryptographic key will be used to control the scrambling. The scrambled coefficients and motion vectors are then subject to video compression before they are transmitted over networks or stored in a storage device.

At the decoder, the compressed video bitstream will first be decompressed (entropy decoding and de-quantization). Authorized users will then use the same key to de-scramble the decompressed coefficients prior to inverse transformation, and de-scramble the motion vectors prior to motion compensation. It can be seen from Fig. 1.1 that the proposed scrambling scheme operates on transformed images. The encryption/decryption operations are very simple to implement. They only

involve randomly flipping the sign and shuffling/rotating blocks of coefficients or motion vectors.

The processing overhead is usually negligible, compared to compression/decompression. In addition, the scrambling is performed prior to compression (quantization and Huffman, run-length, arithmetic, embedded, or other entropy coding.). This generally allows convenient transcoding at the intermediate router, since the decompression (entropy decoding and dequantization) and recoding (or coefficient truncation) do not require the cryptographic key.

It should be noted that this is not exactly the case for the most effective transcoding (sometimes referred to as error-drift-free transcoding) where the error introduced by re-coding the previous frames should be compensated in the following frames. However, in many real applications, error-drift-free transcoding is not implemented (especially in the intermediate routers) due to its extra complexity. Performing scrambling in the frequency domain also makes it easier to control transparency (i.e., what part of the video data is allowed to be freely accessed).

The encryption/decryption operations are designed to preserve, as much as possible, the transformed image properties that allow entropy coders to efficiently compress an image. Although the encryption function and the compression function are illustrated separately in Fig. 1.1, the encryption algorithms will be designed with the compression schemes in mind. Potentially, the compression function can also take advantages of the knowledge of the scrambling scheme to achieve the best possible compression efficiency, although potentially with some other trade-offs.

The design of the scrambling schemes, the potential adverse effect on the compression efficiency can be very well controlled. Aside from easy and secure transcoding, the proposed joint scrambling-compression framework provides some other advantages over those that perform scrambling on the compressed bitstreams.

First, more flexibility is provided to perform selective encryption in our framework. In the frequency domain, it is easier to identify what parts of the data are critical for security purpose. This allows providing different levels of security and transparency. It is also easy to identify what parts of the data are not compressible. For example, the sign bits of the coefficients are usually difficult to compress; yet they are critical for security purpose. This uncompressible data segment can be selected to scramble without affecting the overall compression efficiency.

Some other data segments, such as the motion vector information, are usually losslessly compressed. They therefore can be selected to encrypt without the need to consider the transcoding issue, since it does not make much sense for the transcoder to recode this part of the compressed data. Notice that the selected data can be easily located in the frequency domain without incurring any processing overhead. On the contrary, since the compressed bitstream is usually variable length coded, it is generally difficult to perform fine-scale selective encryption on the compressed bitstream without incurring processing and bit overheads.

Second, encryption using block ciphers such as DES on the compressed bitstreams is more vulnerable to channel errors because a block of 64 bits in DES are boundtogether so that one single bit error in a block will cause the synchronization word/bits contained in that block to be erroneous. The fact that the synchronization information is hidden in the video stream will make it harder to recover from transmission errors in the network.

On the other hand, spatial scrambling in the frequency domain has no adverse impact on the error resiliency. In fact, spatial scrambling of transform coefficients may result in more resiliency to packet loss, due to the fact that the partial information from a spatially local image area may be packetized in different packets, avoiding the loss of an entire local area.

Third, scrambling in the frequency domain may only involve changing the spatial positions of individual frequency coefficients. Thus it may still retain the feasibility of some transformed domain signal processing such as watermarking, and of evaluating some statistical characteristics of the image such as global histogram and energy of the transform coefficients, motion intensity information, etc., without having to access the cryptographic key.

This may be useful in some applications such as multimedia content management and filtering that aim to classify the video data or block certain types of video data. In fact, since the scrambling happens in the frequency domain prior to entropy coding and bitstream formation, the resulting scrambled compressed bitstream will "conform" to the compression format, i.e. a conventional decoder will be able to decode the scrambled compressed bitstream as if it were an ordinary compressed bitstream.

1.2 PROBLEM DEFINITION

Multimedia data security is very important for multimedia commerce on the Internet such as video on- demand and real-time video multicast. The digital video are normally large and during transmission it should be compressed and transmitted.

Traditional cryptographic algorithms/systems for data security are often not fast enough to process the vast amount of data generated by the multimedia applications to meet the real-time constraints. So the encryption technique applied should be efficient for large video data, compression and transmission factors.

This project deals the efficient technique which employs the joint encryption and compression framework to help in compression and transmission on video data. This project consists of four modules. The modules are Selective Bit Scrambling, Block Shuffling, Block Rotation and Security Analysis.

Selective Bit Scrambling scrambles selected bits in the transform coefficients to encrypt an image. Block Shuffling divides each subband into a number of blocks of equal size. Block Rotation improves security with little impact on statistical coding, each block of coefficients can be rotated from encrypted blocks. Block Rotation encrypts the sign bits.

usually add a large amount of processing overhead to meet the real-time video delivery requirement.

2.1.2 PROPOSED SYSTEM

The proposed system includes both scrambling and compression framework in which digital video data is efficiently scrambled in the frequency domain without affecting the compression efficiency significantly.

ADVANTAGES

- ✓ The scrambling process is very simple and efficient.
- ✓ Provides different levels of security, has very limited adverse impact on the compression efficiency and no adverse impact on the error resiliency
- ✓ Allows more flexible selective encryption, transcodability/scalability, and some other useful features.
- ✓ Effectiveness of scrambling motion vector information, which does not seem to have been properly, addressed in prior selective encryption schemes.
- ✓ Encryption using block ciphers such as DES on the compressed bitstreams is more vulnerable to channel errors because a block of 64 bits in DES are bound together so that one single bit error in a block will cause the synchronization word/bits contained in that block to be erroneous.Scrambling in the frequency domain may only involve changing the spatial positions of individual frequency coefficients

CHAPTER 2

LITERATURE REVIEW

2.1 FEASIBLITY STUDY

2.1.1 EXISTING SYSTEM

Traditional cryptographic algorithms/systems for data security are often not fast enough to process the vast amount of data generated by the multimedia applications to meet the real-time constraints.

LIMITATIONS

- ✓ General cryptographic algorithms are not suitable for digital video data.
- ✓ Video scramblers are commonly employed to prevent unauthorized access to video data.
- ✓ These video scrambling systems rely on methods of directly distorting the visual image data and appears unintelligible to a viewer. These scrambling techniques are not efficient for transmitting digital video signals
- ✓ Digital video transmission system usually includes a compression module to reduce transmitted bit rate.
- ✓ Scrambling digital images, the images are first subject to compression, and then the compressed image data is treated as ordinary data and is encrypted/decrypted using traditional cryptographic algorithms such as Digital Encryption Standard (DES). Due to the high data rate of video (even compressed video), they

This proposed system first Opens AVI Video and Extracts frames into BMP images. Then for each image the following steps repeated

- o Read Image data
- o Apply Block Rotation
- o Apply Line Shuffling
- o Apply Sign Encryption

Finally build AVI video from the scrambled images.

2.2 HARDWARE REQUIREMENTS

Processor	:	Intel Processor IV
RAM	:	128 MB
Hard disk	:	20 GB
CD drive	:	40 x Samsung
Floppy drive	:	1.44 MB
Monitor	:	15' Samtron color
Keyboard	:	108 mercury keyboard
Mouse	:	Logitech mouse

2.3 SOFTWARE REQUIREMENTS

Operating System	:	Windows XP/2000
Language used	:	J2sdk1.4.0

CHAPTER 3

DETAILS OF THE METHODOLOGY EMPLOYED

3.1 WAVELET BASED SYSTEMS

We assume the input video frames (original or residual error after motion compensation) are transformed using the wavelet filter banks.It shows 16 subbands that represents a five level wavelet decomposition of an input frame obtained by separable filtering along the vertical and the horizontal directions. Each subband represents selected spatial frequency information of the input video frame. The statistics of the coefficient distribution generally differ from subband to subband. In addition, because the coefficients of the subbands are arranged in the spatial arrangement of the original image, neighboring coefficient correlation exists that can be exploited by a bitstream coder. The goal here is to provide a coefficient scrambling/shuffling method that does not significantly destroy these statistical properties. In this paper, two basic approaches are proposed to scramble the coefficients, each based on the recognition of a different characteristic of the transform coefficient data.

The first one recognizes that although wholesale encryption of individual transform coefficients is generally undesirable (because coefficient encryption adds complexity and destroys the compressibility of the low-entropy coefficient data), some bits of individual transform coefficients have high entropy and can thus be encrypted without greatly affecting compressibility.

The second one recognizes that shuffling the arrangement of coefficients in a transform coefficient map can provide effective security without destroying compressibility, as long as the shuffling does not destroy the low-entropy aspects of the map relied upon by the bitstream coder.

3.1.1 SELECTIVE BIT SCRAMBLING

The first basic approach scrambles selected bits in the transform coefficients to encrypt an image.Each bit of a coefficient can be viewed as one of three types. Significance bits for a coefficient are the most significant bit with a value of 1, and any preceding bits with a value of 0. These bits limit the magnitude of the coefficient to a known range. Refinement bits are the remaining magnitude bits, used to refine the coefficient within the known range. The sign bit determines whether the known range is positive or negative. It is recognized that the efficiency of a bitstream coder such as the bit plane coders proposed in differs, depending on the bit type being coded. Most transforms create a large number of coefficients having small magnitude, and tend to group small magnitude coefficients together. Thus the significance bits have relatively low entropy, and are therefore highly compressible. On the other hand, most transforms produce coefficients with sign bits that have an approximately equal probability of being a 1 or a 0, and that are highly uncorrelated with the sign bits of neighboring coefficients. Refinement bits also tend to have approximately equal probabilities of 1 or 0, and are highly uncorrelated with neighboring refinement bits. Because of their high entropy (and limited predictability), the sign bits and refinement bits are not highly compressible.

The first approach thus selects individual bits that are not highly compressible from each coefficient to scramble. Because these bits have limited predictability to start with, scrambling them results in a negligible decrease in bitstream coding efficiency. For example, we can randomly change the sign of each coefficient. A key-based cryptographically secure pseudo random process controls the sign change process. Note that for many coders, the mean of the image is removed before transform so that the coefficients in the lowpass band are also signed. Because the sign-inverted coefficients distribute their energy over the entire block of pixels they were derived from, sign bit scrambling is quite effective at producing severe degradation in image quality.

The refinement bits of the coefficients can be scrambled too. This does not provide the same level of degradation as sign bit scrambling. Nevertheless, scrambling refinement bits, when done properly, adds an additional level of image degradation and security at low added complexity. A refinement bit scrambler may thus choose to scramble only the most significant, or the two most significant, refinement bits from each coefficient.

3.1.2 BLOCK SHUFFLING

To increase the level of security, block shuffling is proposed. We divide each subband into a number of blocks of equal size. The size of the block can vary for different subbands. Within each subband, blocks of coefficients will be shuffled according to a shuffling table generated using a key. The shuffling table generally will be different for different subbands, and can vary from frame to frame. Since the scrambling performed by block shuffler is block-based, it retains most of the local 2-D statistics of the subband signal. Only coefficients around the block boundary may be slightly affected. Therefore, the negative impact on subsequent statistical coding is very small (e.g., less than 5% bit rate increase while the visual effect of the

shuffling on a decompressed encrypted image is dramatic. Furthermore, a global shuffling of spatially disjoint transform coefficient blocks is much more secure than a local shuffling of coefficients of different frequencies of the same spatial image block. The global block shuffling changes the high-level spatial configuration of the video frequency content, which is much harder for an attacker to analyze than the local shuffling of coefficients of different frequencies as proposed where statistics of different frequency components can be exploited for an efficient attack. In general, block size can be selected to trade security for statistical coding impact, with larger and fewer blocks producing less security but less impact on statistical coding.

3.1.3 BLOCK ROTATION

To further improve security with little impact on statistical coding, each block of coefficients can be rotated to form an encrypted blocks. The encrypted block is selected from a set of eight blocks that are rotated versions of the original block. The key controls the selection process. Block rotation retains most of the local 2-D statistics of the subband signal. It could be considered as a special case of shuffling coefficients within a block taken from certain subband.

3.1.4 SECURITY ANALYSIS

The security of the scrambling process can be analyzed as follows. For the encryption of the sign bits, if a code-breaker is to completely recover a single original frame, an exhaustive search of 2M trials is required, where M is the number of non-zero coefficients in the frame. For a 512x512 frame, assuming, conservatively, that only 256 non-zero coefficients exist, the number of required trials is about 1077.

If an attacker uses a smoothness constraint in the spatial domain to search for the best estimate of the original sign bits, each trial includes an inverse transformation (at least a local inverse transformation), which will make the attack costly. For example, the complexity for a fast n-point inverse DCT is in the order of nlog2n multiplications, which is significant compared to the permutation and substitution operations in DES.

The next step, block shuffling, will render a completely incomprehensible image. Theoretically, it is very difficult to recover the image frame without knowing the shuffling table. Consider a subband that contains 64 blocks. These 64 blocks are shuffled to one of 64! Possible permutations. Note that there may be many blocks that contain only zero coefficients, especially for high frequency subbands that are to be coarsely quantized. For images with large homogeneous regions, it is also possible to have identical blocks.

Assuming there are n zero blocks and all other blocks are different from each other, then the number of different permutations is 64!/n!. If n=48, then the number of different permutations is about 1028, with each permutation requiring the attacker to perform inverse transforms for all blocks affected by the subband permutation. It should be noted that with wavelet transform data, the attacker potentially might try to search for the best estimate directly in the transformed domain by exploiting some structure of the coefficient image such as edge continuity. This attack is, however, difficult to construct due to the diversity of high-level object structure and the uncorrelated nature of the coefficient image, particularly when there is no prior knowledge about the content of the video.

Block rotation further increases the difficulty of recovering an original frame without the key. In this case, assuming eight possible ways of rotation, there are 512 (64x8) potential candidate blocks to fill 64 locations. Again, assuming there are n zero blocks in the decompressed subband and all other blocks are different from each other, then the number of different configurations is 512!/(8n)!, which is significantly larger than 64!/n!.

Depending on the requirements of the application, the proposed three methods can be employed individually or in a combined fashion. If motion compensation is employed in the compression system, we can also apply these three methods to the motion vector field, as will be shown in the following 8x8 DCTbased system. For more secure video transmission, the key can also be updated as time progresses to provide a dynamic key-based scrambling system that would be more secure to the known-plaintext attack.

Note that the random sequence controlling the selective bit encryption and block shuffling process should be generated using a cryptographically secure pseudo-random number generator such as the softwareoptimized fast encryption algorithms.

For the wavelet transform based scrambling system described above, the user can select to leave the low-resolution subbands unscrambled in order to provide some level of transparency. For example, one potential application is to allow free access to low-resolution digital TV signal while requiring a key to watch high definition TV programs. In fact, similar transparency is also possible for the DCT based system to be described where the DC and low frequency components can be left unscrambled.

PERFORMANCE EVALUATION

CHAPTER 4
PERFORMANCE EVALUATION

Testing is a critical element of software quality and assurance and represents the ultimate review of specification design and coding. It is a vital activity that has to be enforced in the development of any system. This could be done in parallel during all the phases of system development. The feedback received from these texts can be used for further enhancement of the system under consideration. The testing phase conducts test using the Software Requirement Specification as a reference and with the goal to see whether system satisfies the specified requirements.

Standard procedures have been followed in testing our system. Test cases are generated for each screen. These test cases will cover every possibility which could result in both positive and negative results. These test plans are maintained for any further testing done on the system.

The main types of tests carried out are:

- Unit Testing.
- Functional Testing.
- Performance Testing.

4.1 UNIT TESTING

1. Test for application window properties.

The properties of the windows are properly aligned an displayed perfectly.

2. Test for mouse operations.

The mouse operations were performed and the necessary operations were fully operational without any exception.

4.2 FUNCTIONAL TESTING

1. Test for various input values in Video Scrambling.

Accurate results were received after execution.

2. Test for various input values in Video Descrambling.

Accurate results were received after execution.

4.3 PERFORMANCE TESTING

1. This is required to assure that an application performed adequately, has the capability to handle any workload, delivers its results in expected time and uses an acceptable level of resource and it is an aspect of operational management.

Results after operations handled large input values and produced accurate results in expected time.

CHAPTER 5
CONCLUSION

The unique feature of multimedia data is that the data rate is high while the information value is usually lower than ordinary electronic information. This feature justifies the employment of lightweight cryptographic algorithms to reduce the computational overhead while still maintaining reasonable level of security. We show in this paper that by jointly considering encryption and compression, efficient compression-friendly digital video scrambling techniques can be designed to facilitate simple implementation and to allow for video transcodability/scalability, transparency, and other useful encryption-domain signal processing. The proposed scrambling techniques appear to achieve a very good compromise between several desirable properties such as speed, security, file size and transcodability, therefore is very suitable for network video applications.

FUTURE ENHANCEMENTS

CHAPTER 6
FUTURE ENHANCEMENTS

The project can be further enhanced by increasing the support for more video formats. This will make the project to be applicable in various situations.

The other enhancement that can be developed is the support for sound along with video. The inclusion of the sound will make project more efficient. Further the sound can also be scrambled which increases the security and privacy.

The user interfaces may also be further developed and made more attractive with increased options and Toolbar can also be included in the user interface.

The player can be further enhanced by the inclusion of more controls and the formats that are being supported can also be increased which results in the ability to watch various video formats.

And other enhancements like the user defined keys for encryption can also be applied along with the support for multiple clients which will enable the access of the file from multiple nodes.

APPENDIX

CHAPTER 7
APPENDIX

7.1 APPENDIX1-SOURCE CODE

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.FilenameFilter;
import java.util.Arrays;
import org.jim2mov.test.Files2MovExample;
import org.jim2mov.core.MovieInfoProvider;

public class Video_Scrambling extends JFrame implements ActionListener{

    private Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    //Getting the User's Screen Dimensions.
    private JMenuBar menuBar;
    private JMenu menu_File,menu_View,menu_Process;
    private JMenuItem menuItem;
    private JDesktopPane desktop;
    private int count = 0;
    Object object[] = new Object[3] ;

    public Video_Scrambling(){

        super("Video Scrambling");
        setIconImage(getToolkit().GetImage("Images/JESUS_CHRIST.gif"));
        //Setting the Program's Icon.
        setSize(d.width-20,d.height-40);    //Setting Main Window Size.
        setLocation(10,10);                 //Setting Main Window
                                                    Location.

        this.setJMenuBar(createMenuBar());  //adding menubar to frame


        desktop=new JDesktopPane();
```

```
            setContentPane(desktop);

            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }//Video_Scrambling constructor ends

        public JMenuBar createMenuBar(){

            menuBar = new JMenuBar();

            menu_File = new JMenu("File");
            menu_File.setMnemonic((int)'F');  // adding mnemonics to menu
            menu_File.add(createMenuItem("Login",'L'));
            menu_File.add(createMenuItem("Exit",'E'));
            menuBar.add(menu_File);           //  adding menu's to MenuBar

            object[0] = menu_File;

            menu_View = new JMenu("View");
            menu_View.setMnemonic((int)'I');
            menu_View.add(createMenuItem("Video",'V'));
            menu_View.add(createMenuItem("EXtracted Image",'X'));
            menu_View.add(createMenuItem("Scrambled Image",'S'));
            menu_View.add(createMenuItem("Play SCrambled video",'C'));
            menuBar.add(menu_View);

            object[1] = menu_View;

            menu_Process = new JMenu("Process");
            menu_Process.setMnemonic((int)'P');
            menu_Process.add(createMenuItem("ExTract",'T'));
            menu_Process.add(createMenuItem("ScRamble",'R'));
            menu_Process.add(createMenuItem("Create Video",'A'));
            menuBar.add(menu_Process);

            object[2] = menu_Process;

            return menuBar;
        }//createMenuBar method ends

        public JMenuItem createMenuItem(String title,char key){

            menuItem = new JMenuItem(title);
            menuItem.setMnemonic(key);
            menuItem.setAccelerator(KeyStroke.getKeyStroke(key,2));
            menuItem.addActionListener(this);

            if(!(title.equals("Login")|title.equals("Exit")))
                menuItem.setEnabled(false);
                return menuItem;

        }//createMenuItem method ends

        // Display method to add internalFrame to the desktopPane
        void display(JInternalFrame obj) {

            new CenterFrame(obj); // to place the InternalFrame in the
center
            desktop.add(obj);
            try {
                obj.setSelected(true);
            }
            catch(java.beans.PropertyVetoException e2){}
        }


        public void actionPerformed(ActionEvent ae){

            JMenuItem ite = (JMenuItem)(ae.getSource());
            String obj = ite.getText();


            if(obj.equals("Login")){

                //Creating Object 4 Login.java & calling the same
                Login objLogin = new Login(object);
                display(objLogin);

            }//if Login ends


            if(obj.equals("Exit")){

                System.exit(0);
            }//if Exit ends


            if(obj.equals("Video")){

                //Creating Object 4 availableVideo.java & calling the
same
                availableVideo objavailableVideo = new
availableVideo(object,desktop);
                display(objavailableVideo);
            }//if Video ends


            if(obj.equals("ExTract")){

                JMenuItem mi_Extract
=(JMenuItem)((JMenu)object[2]).getItem(0);
                mi_Extract.setEnabled(false);
                //creating Object 4 textarea.java & calling the same
                textarea objtextArea=new textarea();    //class to diplay
the

processing

                //creating Object 4 FrameAccess.java & calling the same
                FrameAccess objFrameAccess=new FrameAccess();
//class to extract image's from video

        objFrameAccess.open(availableVideo.videoname,objtextArea.txtaMe
ssage,object,objtextArea);
                display(objtextArea);

            }//if ExTract ends


            if(obj.equals("EXtracted Image")){

                File file = new File("data/extracted/");

                //To read the files inside the Image folder and store it in a
string array
                FilenameFilter only = new OnlyExt("bmp");
                String image[] = file.list(only);

                int file_names[] = new int[image.length];

                for(count = 0; count < image.length; count++){
                    String sp[]=image[count].split(".bmp");
                    file_names[count]=Integer.parseInt((sp[0].trim()));
                }//for loop ends
                 //To arrange the files in ascending order
                 Arrays.sort(file_names);
                 for(count = 0; count < file_names.length; count++){

        image[count]="data/extracted/"+file_names[count]+".bmp";
                 }

                 ImageView objImageView = new
ImageView(image,((JMenu)object[1]).getItem(1));
                 display(objImageView);

            }//if EXtracted Image ends


            if(obj.equals("ScRamble")){

                File file = new File("data/extracted/");

                //To read the files inside the Image folder and store it in a
string array
                FilenameFilter only = new OnlyExt("bmp");
                String image[] = file.list(only);

                //Creating Object 4 Scrambling.java & calling the same
```

```
                Scrambling objScrambling = new
Scrambling(image.length,object);
                display(objScrambling);

            }//if Scramble ends


            if(obj.equals("Scrambled Image")){

                File file = new File("data/scrambled/");

                //To read the files inside the Image folder and store it in a
string array
                FilenameFilter only = new OnlyExt("bmp");
                String image[] = file.list(only);

                int file_names[] = new int[image.length];

                for(count = 0; count < image.length; count++){
                    String sp[]=image[count].split(".bmp");
                    file_names[count]=Integer.parseInt((sp[0].trim()));
                }//for loop ends
                 //To arrange the files in ascending order
                 Arrays.sort(file_names);
                 for(count = 0; count < file_names.length; count++){

      image[count]="data/scrambled/"+file_names[count]+".bmp";
                 }

                 ImageView objImageView = new
ImageView(image,((JMenu)object[1]).getItem(2));
        display(objImageView);

            }//if Scrambled Image ends

            if(obj.equals("Create Video")){

                JMenuItem mi_Video =
(JMenuItem)((JMenu)object[2]).getItem(2);
                mi_Video.setEnabled(false);
```

```
                File file = new File("data/scrambled/");

                //To read the files inside the Image folder and store it in a
string array
                FilenameFilter only = new OnlyExt("bmp");
                String image[] = file.list(only);
                for(count = 0; count < image.length; count++){}//for
loop ends

                String arg[] = new String[count];
                JOptionPane.showMessageDialog(null,"Started Creation
of Video File","Information",1);
                for(int i=0 ; i<count ;i++)
                {
                        arg[i] = "data/scrambled/"+(i+1)+".bmp";
            }

                //calling Files2MovExample.java
                Files2MovExample objFiles2MovExample = new
Files2MovExample(arg,MovieInfoProvider.TYPE_QUICKTIME_JPEG,
"data/scrambled_video/Scrambled_video.mov");

                if(objFiles2MovExample.cond())
                {
                    JOptionPane.showMessageDialog(null,"Created
Video File","Information",1);
                    JMenuItem mi_PlayVideo =
(JMenuItem)((JMenu)object[1]).getItem(3);
                    mi_PlayVideo.setEnabled(true);

          //calling Server.java
                    new Server();
                }


            }//if Create Video ends

            if(obj.equals("Play SCrambled Video")){
```

```
                JInternalFrame f = new JInternalFrame("JMF Video
Player",true,true,true,true);
                Container frameCP = f.getContentPane();
                JMFPlayer p = new
JMFPlayer(f,"data/scrambled_video/Scrambled_video.mov");
                frameCP.add(BorderLayout.CENTER, p);
                f.setSize(200, 200);
                f.setVisible(true);
                display(f);

            }//if Play Scrambled Video ends


        }//actionPerformed method ends


        public static void deleteFiles(){

            try {

          String
file_location[]={"data\\extracted\\","data\\scrambled\\","data\\scrambled_vid
eo\\"};

                int count = 0;

                for(count = 0;count < file_location.length; count++){

                 File f = new File(file_location[count]);

                 String files[] = f.list();

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

                        File del_file = new
File(file_location[count]+files[i]);

                        del_file.delete();
```

```
                    }
                    }
                }
                 catch(Exception e) {

                        System.err.println("Error in deleteFiles method in
Video_Scrambling.java "+e);
                        e.printStackTrace();
                }
            }//method deleteFiles ends


          public static void main(String[] args){

        //Make sure we have nice window decorations.
        JFrame.setDefaultLookAndFeelDecorated(true);

        deleteFiles();      //call to deleteFiles method
                new Video_Scrambling().show();

          }//void main method ends

}//Main class ends
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.FilenameFilter;
import java.util.Arrays;
import org.jim2mov.test.Files2MovExample;
import org.jim2mov.core.MovieInfoProvider;

public class Video_Descrambling extends JFrame implements
ActionListener{
```

```
        private Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
        //Getting the User's Screen Dimensions.
        private JMenuBar menuBar;
        private JMenu menu_File,menu_View,menu_Process;
        private JMenuItem menuItem;
        private JDesktopPane desktop;
        private int count = 0;
        Object object[] = new Object[3] ;

        public Video_Descrambling(){

                super("Video Descrambling");
                setIconImage (getToolkit().getImage
("Images/JESUS_CHRIST.gif"));       //Setting the Program's Icon.
                setSize(d.width-20,d.height-40); //Setting Main Window Size.
                setLocation(10,10);                  //Setting Main Window
Location.

                this.setJMenuBar(createMenuBar());     //adding menubar to
frame


                desktop=new JDesktopPane();

                setContentPane(desktop);

                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }//Video_Descrambling constructor ends


        public JMenuBar createMenuBar(){

                menuBar = new JMenuBar();

                menu_File = new JMenu("File");
                menu_File.setMnemonic((int)'F');     // adding mnemonics to
menu
                menu_File.add(createMenuItem("Login",'L'));
                menu_File.add(createMenuItem("Exit",'E'));
                menuBar.add(menu_File);                 //  adding menu's to
MenuBar

                object[0] = menu_File;

                menu_View = new JMenu("View");
                menu_View.setMnemonic((int)'I');
                menu_View.add(createMenuItem("Video",'V'));
                menu_View.add(createMenuItem("EXtracted Image",'X'));
                menu_View.add(createMenuItem("DeScrambled Image",'S'));
                menu_View.add(createMenuItem("Play DesCrambled
video",'C'));
                menuBar.add(menu_View);

                object[1] = menu_View;

                menu_Process = new JMenu("Process");
                menu_Process.setMnemonic((int)'P');
                menu_Process.add(createMenuItem("ExTract",'T'));
                menu_Process.add(createMenuItem("Descramble",'D'));
                menu_Process.add(createMenuItem("Create Video",'A'));
                menuBar.add(menu_Process);

                object[2] = menu_Process;

                return menuBar;
        }//createMenuBar method ends


        public JMenuItem createMenuItem(String title,char key){

                menuItem = new JMenuItem(title);
                menuItem.setMnemonic(key);
                menuItem.setAccelerator(KeyStroke.getKeyStroke(key,2));
                menuItem.addActionListener(this);

                if(!(title.equals("Login")|title.equals("Exit")))
                        menuItem.setEnabled(false);
                return menuItem;
```

```
        }//createMenuItem method ends

        // Display method to add internalFrame to the desktopPane
        void display(JInternalFrame obj) {

                new CenterFrame(obj); // to place the InternalFrame in the
center
                desktop.add(obj);
                try {
                        obj.setSelected(true);
                }
                catch(java.beans.PropertyVetoException e2){}
        }


        public void actionPerformed(ActionEvent ae){

                JMenuItem ite = (JMenuItem)(ae.getSource());
                String obj = ite.getText();


                if(obj.equals("Login")){

                        //Creating Object 4 Login.java & calling the same
                        Login objLogin = new Login(object);
                        display(objLogin);

                }//if Login ends


                if(obj.equals("Exit")){

                        System.exit(0);
                }//if Exit ends


                if(obj.equals("Video")){
```

```
                        //Creating Object 4 availableVideo.java & calling the
same
                        availableVideo objavailableVideo = new
availableVideo(object,desktop);
                        display(objavailableVideo);
                }//if Video ends


                if(obj.equals("ExTract")){

                        JMenuItem mi_Extract =
(JMenuItem)((JMenu)object[2]).getItem(0);
                        mi_Extract.setEnabled(false);
                        //creating Object 4 textarea.java & calling the same
                        textarea objtextArea=new textarea();  //class to diplay the
processing

                        //creating Object 4 FrameAccess.java & calling the same
                        FrameAccess objFrameAccess=new FrameAccess();
//class to extract image's from video

                objFrameAccess.open(availableVideo.videoname,objtextArea.txtaMe
ssage,object,objtextArea);
                        display(objtextArea);

                }//if ExTract ends


                if(obj.equals("EXtracted Image")){

                        File file = new File("data/extracted/");

                        //To read the files inside the Image folder and store it in a
string array
                        FilenameFilter only = new OnlyExt("bmp");
                        String image[] = file.list(only);

                        int file_names[] = new int[image.length];

                        for(count = 0; count < image.length; count++){
```

```
                    String sp[]=image[count].split(".bmp");
                    file_names[count]=Integer.parseInt((sp[0].trim()));
                }//for loop ends
                //To arrange the files in ascending order
                Arrays.sort(file_names);
                for(count = 0; count < file_names.length; count++){

    image[count]="data/extracted/"+file_names[count]+".bmp";
                }

                ImageView objImageView = new
ImageView(image,((JMenu)object[1]).getItem(1));
                display(objImageView);

            }//if EXtracted Image ends


            if(obj.equals("Descramble")){

                File file = new File("data/extracted/");

                //To read the files inside the Image folder and store it in a
string array
                FilenameFilter only = new OnlyExt("bmp");
                String image[] = file.list(only);

                //Creating Object 4 Descrambling.java & calling the
same
              Descrambling objDescrambling = new
Descrambling(image.length,object);
              display(objDescrambling);

            }//if Scramble ends


            if(obj.equals("DeScrambled Image")){

                File file = new File("data/descrambled/");
```

```
                //To read the files inside the Image folder and store it in a
string array
                FilenameFilter only = new OnlyExt("bmp");
                String image[] = file.list(only);

                int file_names[] = new int[image.length];

                for(count = 0; count < image.length; count++){
                    String sp[]=image[count].split(".bmp");
                    file_names[count]=Integer.parseInt((sp[0].trim()));
                }//for loop ends
                //To arrange the files in ascending order
                Arrays.sort(file_names);
                for(count = 0; count < file_names.length; count++){

    image[count]="data/descrambled/"+file_names[count]+".bmp";
                }

                ImageView objImageView = new
ImageView(image,((JMenu)object[1]).getItem(2));
              display(objImageView);

            }//if Scrambled Image ends

            if(obj.equals("Create Video")){

                JMenuItem mi_Video =
(JMenuItem)((JMenu)object[2]).getItem(2);
                mi_Video.setEnabled(false);

                File file = new File("data/descrambled/");

                //To read the files inside the Image folder and store it in a
string array
                FilenameFilter only = new OnlyExt("bmp");
                String image[] = file.list(only);
                for(count = 0; count < image.length; count++){}//for
loop ends

                String arg[] = new String[count];
```

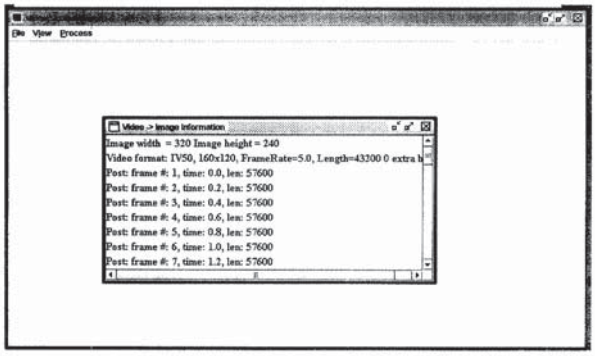
```
                JOptionPane.showMessageDialog(null,"Started Creation
of Video File","Information",1);
                for(int i=0 ; i<count ;i++)
                {
                                arg[i] =
"data/descrambled/"+(i+1)+".bmp";
            }

                //calling Files2MovExample.java
                Files2MovExample objFiles2MovExample = new
Files2MovExample(arg,MovieInfoProvider.TYPE_QUICKTIME_JPEG,
"data/video/Descrambled_video.mov");

                if(objFiles2MovExample.cond())
                {
                    JOptionPane.showMessageDialog(null,"Created
Video File","Information",1);
                    JMenuItem mi_PlayVideo =
(JMenuItem)((JMenu)object[1]).getItem(3);
                  mi_PlayVideo.setEnabled(true);
                }



            }//if Create Video ends

            if(obj.equals("Play DesCrambled Video")){

                JInternalFrame f = new JInternalFrame("JMF Video
Player",true,true,true,true);
                Container frameCP = f.getContentPane();
                JMFPlayer p = new
JMFPlayer(f,"data/video/Descrambled_video.mov");
                frameCP.add(BorderLayout.CENTER, p);
                f.setSize(200, 200);
                f.setVisible(true);
                display(f);

            }//if Play Scrambled Video ends
```

```
        }//actionPerformed method ends


        public static void deleteFiles(){

            try {

          String
file_location[]={"data\\descrambled\\","data\\video\\","data\\extracted\\","da
ta\\scrambled_video\\"};

                int count = 0;

                for(count = 0;count < file_location.length; count++){

                  File f = new File(file_location[count]);

                  String files[] = f.list();

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

                      File del_file = new
File(file_location[count]+files[i]);

                      del_file.delete();

                  }
                  }
              }
              catch(Exception e) {

                  System.err.println("Error in deleteFiles method in
Video_Descrambling.java "+e);
                  e.printStackTrace();
              }
        }//method deleteFiles ends
```

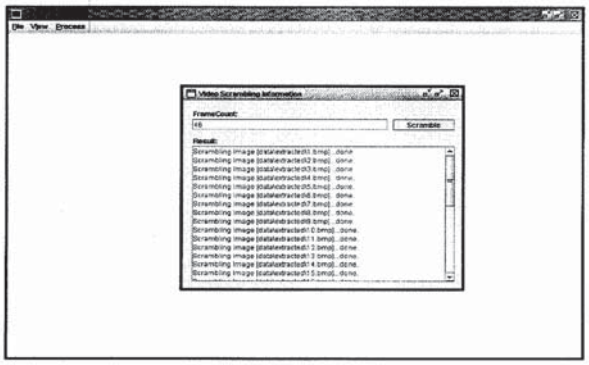
```
    public static void main(String[] args){

    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    deleteFiles();      //call to deleteFiles method
            new Video_Descrambling().show();

     }//void main method ends

}//Main class ends
```

7.2 APPENDIX2- SCREEN SHOTS

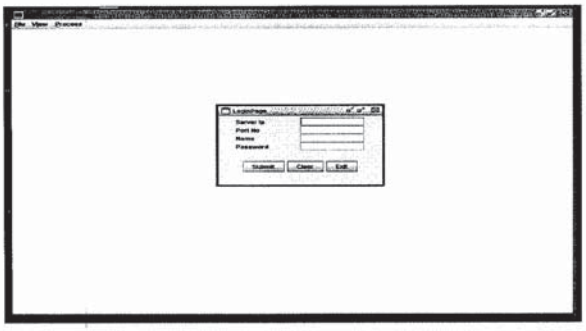
7.2.1 VIDEO SCRAMBLING LOGIN



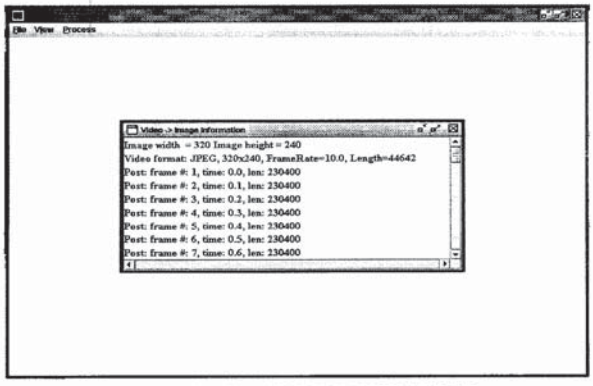
7.2.2 VIDEO EXTRACTION FOR SCRAMBLING



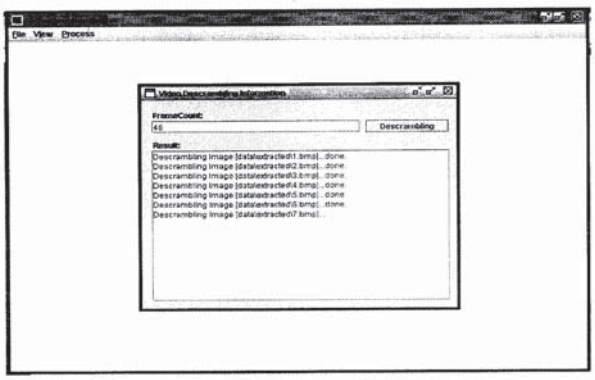
7.2.3 IMAGE CONVERSION FOR SCRAMBLING



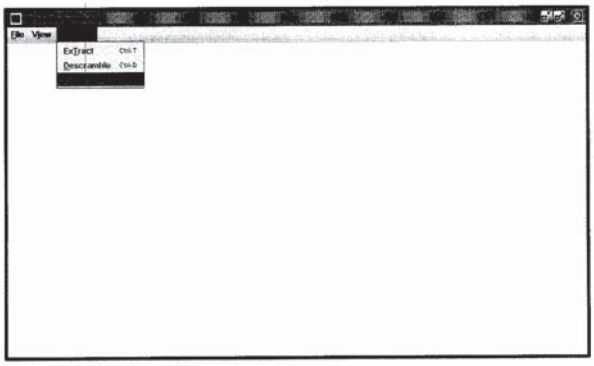
7.2.4 VIDEO DESCRAMBLING LOGIN



7.2.5 VIDEO EXTRACTION FOR DESCRAMBLING



7.2.6 IMAGE CONVERSION FOR DESCRAMBLING



7.2.7 ORIGINAL VIDEO FILE RECREATION

REFERENCES

REFERENCES

PRINT MEDIA

1. Mathew Thomas , **" A tour of Java Swing – Guide"**, PHI , 2000.

2. M C Bridge , 1" Image Compression ", Pearson Education, Edition 2000.

3. Patrick Naughton , **" Complete Reference Java "**, Tata McGraw Hill , Edition 2001.

4. Roger.S.Pressman **" Software Engineering A Practioners Approach ",**Tata McGraw Hill, Edition 2001.

ONLINE SOURCES

1. http://java.sun.com/

2. http://java.sun.com.docs

3. http://developer.java.sun.com

4. http://java.sun.com/products/