P-1823

VIDEO SCRAMBLING

A PROJECT REPORT

Submitted by

S.ARUN	**71203104003**
N.PREMNATHHAN	**71203104025**
P.SHANKARANARAYANAN	**71203104303**

In partial fulfillment for award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2007

P-1823

DECLARATION

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report **"Video Scrambling"** is the bonafide work of "S.Arun (71203104003), N.Premnathhan (71203104025) and P.Shankaranarayanan (71203104303)", who carried out the project work under my supervision.

(Signature)	(Signature)
Dr.S.Thangasamy	Mr.S.Mohanavel
	SUPERVISOR
Head of the Department	Senior Lecturer
Dept. of Computer Science & Engg.	Dept. of Computer Science &Engg.
Kumaraguru College of Technology	Kumaraguru College of Technology
Chinnavedampatti PO	Chinnavedampatti PO
Coimbatore-641006.	Coimbatore-641006.

Submitted for Viva Voce Examination held on 25.04.07

Internal Examiner 25.4

External Examiner 25/4/07

DECLARATION

We hereby declare that the project entitled **"Video Scrambling"** is a record of the original work done by us and to the best of our knowledge.

The report is submitted in partial fulfillment of the requirements for the award for the Degree of Bachelor of Engineering in Computer Science and Engineering of Anna University, Chennai.

Place: Coimbatore.
Date: 23.04.2007

(S.Arun)

(N.Premnathhan)

(P.Shankaranarayanan)

ACKNOWLEDGEMENT

We would like to extend our gratitude to **Dr.Joseph.V.Thanikal,** the Principal of our college, for providing us the required resources to proceed with our project.

We would like to express our sincere thanks to **Dr.S.Thangasamy,** Head of the Department, Department of Computer Science and Engineering, Kumaraguru College of Technology for his guidelines that motivated us to develop a good product.

We are grateful to **Mrs.P.Devaki,** Assistant Professor, Department of Computer Science and Engineering, Kumaraguru College of Technology for her encouragement and support at various levels of this project work.

We also express our sincere thanks to **Ms.S.Rajini,** Senior Lecturer, Project Coordinator, Department of Computer Science and Engineering, Kumaraguru College of Technology for her valuable guidance and encouragement at every stage of this project.

We take immense pleasure in expressing our heartfelt thanks to our guide, **Mr.S.Mohanavel,** Senior Lecturer, Department of Computer Science and Engineering, Kumaraguru College of Technology for his inspiration and guidance throughout this project. We are very grateful to him for his help rendered to us in handling various tough spots during this project.

Most of all, we thank our parents and friends for their blessings, help and support without which we would not be able to do anything.

ABSTRACT

This project entitled 'Video Scrambling' presents a 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.

The project consists of four main 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. Selective Bit Scrambling scrambles selected bits in the transform coefficients to encrypt an image. Bock 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 to form an encrypted blocks. Block Rotation encrypts the sign bits.

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

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

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

JPEG	-	Joint Photographic Experts Group
BMP	-	Bit Map Image
IPR	-	Intellectual Property Right
DVD	-	Digital Versatile Disk
MPEG	-	Motion Pictures Experts Group
DES	-	Digital Encryption Standard
PDA	-	Portable Desktop 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

INTRODUCTION

CHAPTER 1
INTRODUCTION

Use of digital media has exploded in the past few years, primarily due to several distinct advantages that digital media can offer over analog media. These advantages include higher quality, easier editing, perfect copying and easier and more efficient transmission over information network. The wide dissemination of digital media also creates some potential problems. Due to the popularity of Internet commerce and digital library applications, the intellectual property right (IPR) protection is becoming increasingly important.

Content providers will be reluctant to provide their valuable contents if they are not assured that their contents are securely protected. Some good examples are the deployments of the digital versatile disk (DVD) market and the online music market. The IPR management and protection issue is currently being addressed in the emerging MPEG-4 standards for moving pictures compression.

Several technologies have been developed for IPR protection. One is conditional access through encryption. The digital media will be scrambled before it is distributed. Only authorized users who have the proper key for decryption can access the clear content. The other one is digital watermarking that securely embeds hidden message into the multimedia data to identify the owner, or the buyer of a digital media. These two techniques are complementary to each other. We focus on conditional access through encryption in this project.

Digital images/video are often communicated or distributed over non-private channels, such as satellite links, cable television networks, wireless networks, and the Internet. Conditional access systems for private digital image/video transmission or storage are a necessity for many applications, for example, pay-TV, confidential videoconferences, confidential facsimile transmissions, and medical image transmission and storage in a database.

In general, complex cryptography techniques make cracking of the system difficult, but are also expensive to implement. Since digital video transmission system usually includes a compression module that aims to reduce the transmitted bit rate, the cryptography techniques have to be carefully designed to avoid potential adverse impact on the compression efficiency, and on the functionalities that the compression format provides.

Video scramblers are commonly employed to prevent unauthorized access to video data. Several video scrambling systems rely on methods of directly distorting the visual image data (in the spatial domain) such that, without de-scrambling, the video appears unintelligible to a viewer. These scrambling techniques are not efficient for transmitting digital video signals because they, in general, will significantly change the statistical property of the original video signal, thus making it very difficult to compress. Another potential drawback of scrambling video in the spatial domain is that the highly spatially- and temporally- correlated nature of the video data can be used for efficient attacks.

In many systems for 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 usually add a large amount of

processing overhead to meet the real-time video delivery requirement. Note that for larger sized compressed video bit streams such as those in the digital cinema application, real-time software-based decompression itself still remains a very challenging problem. This is also true for applications running on low powered handheld devices such as PDA and videophones. To reduce the amount of processing overhead, selective encryption of the MPEG compressed video data has been proposed.

For example, in selective encryption, only the entropy-coded I frames, or the entropy-coded I frames and Intra-coded blocks of predictive (P/B) frames may be encrypted. However in some cases the encryption of I frames alone does not provide sufficient security, due to the leakage of partial information through I-blocks and P-blocks with high energy in P/B frames. They suggested increasing the frequency of I frames. The increase of security, however, is achieved at the expense of reducing the compression efficiency and increasing the computational complexity for encryption and decryption. In fact increasing the frequency of I frames may not resolve the problem, since a smart attacker can simply skip the I frames and use the frames preceding the I frames as the reference frames to reconstruct the frames immediately following the I frames.

In this way, the partial information available in the P/B residue frames will accumulate and eventually become comprehensible. A more secure MPEG encryption algorithm is described in that encodes half of the data using DES and has a 47% gain in terms of number of XOR operations over DES. Although it may achieve real-time encryption/decryption for low-resolution, low bit rate MPEG sequence, it will have real-time implementation problem for high resolution, and high bit rate sequences.

The above mentioned systems of scrambling the compressed bitstream may also be vulnerable to possible plain text attacks that take advantage of the syntax of the compression algorithm such as the known synchronization word or End of Block symbol that are often used to limit error propagation. To selectively encrypt some segments of the compressed data such as Intra-coded blocks often incurs additional header overhead to locate such segments. In addition, in some applications, a scrambled compressed video has to be transcoded, e.g., from a high bit rate to a low bit rate, due to change of the nature of the transmission channels.

The above mentioned classical approach is subject to security threat for transcoding at intermediate routers of the transmission channel because the scrambling key has to be available at the transcoding points in order to decrypt, decompress, recompress and re-encrypt. This imposes a big challenge to the key management system and the intermediate routers have to be decryption/encryption capable. In some other applications, it is desirable that the encryption allows the content to be somewhat transparent instead of totally indiscernible. For example, a roadcaster of pay-TV may want to promote a contract with nonpaying watcher, by allowing them to access some of the video information. Classical approaches give no transparency.

In fact, one of the distinctions between multimedia data and ordinary electronic information such as bank information and classified documents is that the value of the multimedia information is much lower, while the bit rate is much higher. Some portions of the multimedia data may not be as significant as the other portions, thereby allowing lower security. On the other hand, the complexity involved for encrypting the high rate multimedia data is of concern, especially for real-time software-based interactive

applications that run on low powered devices. Some other features such as allowing transparency and allowing easy and secure transcoding without having to access the cryptographic key may also be desirable.

To address the complexity issue, it is decided to shuffle the DCT coefficients within an 8x8 block for JPEG/MPEG based transmission system. This technique, although very simple to implement, changes the statistical property (run-length characteristic) of the DCT coefficients. As a result, it may increase the bit rate of the compressed video by as much as 50%. It has also been shown that this cipher is vulnerable to frequency-based attack that exploits the property that the non-zero AC coefficients have the tendency to appear earlier in the zigzag order. This approach also may not be amenable to secure transcoding.

Another frequency domain scrambling scheme was proposed in where lines of wavelet coefficients are permuted. Although line shuffling preserves some one-dimensional statistical property of the coefficient image, it destroys the two-dimensional statistical property and thus may potentially affect the compression efficiency significantly. In this paper, we propose a joint scrambling and compression framework in which digital video data are efficiently scrambled in the frequency domain without affecting the compression efficiency significantly.

The proposed framework has the nice feature that the scrambling process is very simple and efficient. It provides different levels of security, has very limited adverse impact on the compression efficiency and no adverse impact on the error resiliency, and allows more flexible selective encryption, transcodability/scalability, transparency and some other useful features. We will also illustrate the importance and effectiveness of

scrambling motion vector information, which does not seem to have been properly addressed in prior selective encryption schemes.

1.1 OVERVIEW OF THE PROPOSED FRAMEWORK

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, transcoding/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.

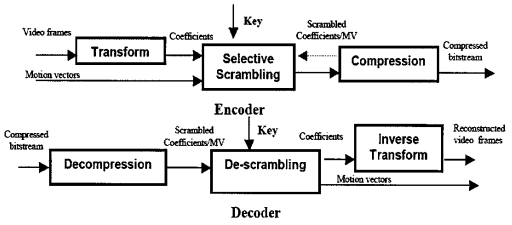


FIG 1.1 GENERAL ARCHITECTURE OF THE PROPOSED SCRAMBLING SYSTEM

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.

DESCRIPTION OF THE PROBLEM

CHAPTER 2
DESCRIPTION OF THE PROBLEM

The digital video encryption is required for most E – commerce and multimedia applications. The digital video data are normally large and during transmission it should be compressed and transmitted. So the encryption technique applied should be efficient for large video data, compression and transmission factors. This project deals the efficient technique which helps in compression and transmission on video data.

2.1 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.

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

2.2 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. The proposed framework has the nice feature that

- 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, transparency 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
- This proposed system first Opens AVI Video and Extracts frames into BMP images. Then for each image the following steps repeated
    i. Read Image data
    ii. Apply Block Rotation
    iii. Apply Line Shuffling
    iv. Apply Sign Encryption

  Finally build AVI video from the scrambled images.

2.3 SYSTEM ENVIRONMENT

The front end is designed and executed with the J2SDK1.4.0 handling the core java part with User interface Swing component. Java is robust, object oriented, multi-threaded, distributed, secure and platform independent language. It has wide variety of package to implement our requirement and number of classes and methods can be utilized for programming purpose. These features make the programmer's to implement to require concept and algorithm very easier way in Java.

The features of Java as follows:

- Core java contains the concepts like Exception handling, Multithreading; Streams can be well utilized in the project environment.
- The Exception handling can be done with predefined exception and has provision for writing custom exception for our application.

- Garbage collection is done automatically, so that it is very secure in memory management.
- The user interface can be done with the Abstract Window tool Kit and also Swing class. This has variety of classes for components and containers. We can make instance of these classes and this instances denotes particular object that can be utilized in our program.
- Event handling can be performed with Delegate Event model. The objects are assigned to the Listener that observe for event, when the event takes place the corresponding methods to handle that event will be called by Listener which is in the form of interfaces and executed.
- This application makes use of ActionListener interface and the event click event gets handled by this. The separate method actionPerformed() method contains details about the response of event.
- Java also contains concepts like Remote method invocation; Networking can be useful in distributed environment.

2.4 SYSTEM REQUIREMENTS

HARDWARE SPECIFICATIONS:

Processor	:	Intel Processor
RAM	:	256 MB
Hard Disk	:	40 GB
CD Drive	:	40X CD-ROM Drive
Floppy Drive	:	1.44 MB FDD
Monitor	:	15" Color Monitor
Keyboard	:	108 Keys Keyboard
Mouse	:	Scroll Mouse

SOFTWARE SPECIFICATIONS:

Operating System	:	Windows XP/2000
Languages Used	:	J2SDK 1.4.0

CHAPTER 3
SYSTEMS ANALYSIS

System analysis can be defined, as a method that is determined to use the resources, machine in the best manner and perform tasks to meet the information needs of an organization.

3.1 SYSTEM DESCRIPTION

The main process done here is scrambling of the video file and descrambling of the video file .First the required video file is being selected. Then the particular video file is extracted into frames with the specified time limit, this is extracted as JPEG format. Then these are being converted into the BMP format. Then the scrambling of the video file is done where the block shuffling and the block rotation is done. The recreation of the video file is done with these frames. The scrambled video is sent through the network.

At the receivers end the descrambling is being done. Here the scrambled video file is being extracted into frames with the same time limit used in the scrambling process. These extracted frames will be in the JPEG format. We convert these JPEG images into BMP images. The descrambling is done by the reverse process of block rotation and block shuffling. With these frames video file is created which is the descrambled, original video file.

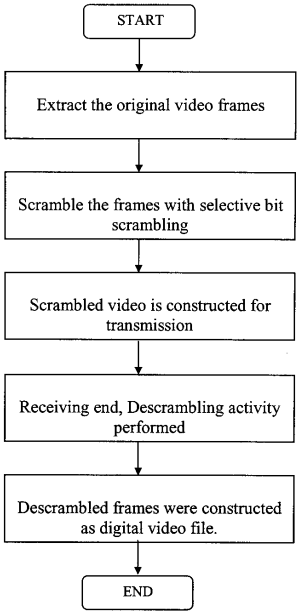


FIG 3.1 VIDEO SCRAMBLING AND DESCRAMBLING PROCESS

3.2 PROPOSED SYSTEM

The proposed system follows wavelet transform based compression. Here 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 represent 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.

3.2.1 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 low pass 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.

3.2.2 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.

3.2.3 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.

3.2.4 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 DCT based 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.

CHAPTER 4
SYSTEM DESIGN

Design is concerned with identifying software components specifying relationships among components. Specifying software structure and providing blue print for the document phase.

Modularity is one of the desirable properties of large systems. It implies that the system is divided into several parts. In such a manner, the interaction between parts is minimal clearly specified.

Design will explain software components in detail. This will help the implementation of the system. Moreover, this will guide the further changes in the system to satisfy the future requirements.

4.1 INPUT DESIGN

Form design is a tool with a message; it is the physical carrier of the data or information. This system deals with the video data in the form of avi file format. These files are stored inside the data folder. The data folder contains the video folder with the original video data, extracted folder with bitmap file. Each frame is extracted as a bitmap file and stored in this folder. The scrambled folder contains scrambled bitmap files and the descrambled folder contains the decrypted bitmap files. The entire four folders will be used during execution of project. The Java interface retrieves the original video and split that into frames. The option is available in the form as follows.

The file menu consists of the LOGIN and the EXIT submenus. When clicking on the login submenu it prompts us to enter the NAME and the

PASSWORD. When given promptly is authenticates. These user name and password are stored in a separate database. The other two menus are the VIEW and the PROCESS. Under the PROCESS menu, EXTRACT, SCRAMBLE and CREATE VIDEO are the submenus. The view is used to view the original video file and the scrambled video file. Now after selecting the video file we select the EXTRACT option. This extracts the video file into the JPEG format into the scrambling folder. Then we select the scramble process. Here the frame count is being specified. This represents the number of frames to which has to be converted for creating the scrambling video. After the scrambling is done we select the CREATE VIDEO option so that the scrambled video file is created.

The descrambling interface has similar menus and submenus instead of SCRAMBLE submenu we have DESCRAMBLE SUBMENU. But the login screen is different. This consists of similar login name and password to that of scrambling interface also add up with the options of SERVER IP ADDRESS and the PORT NUMBER. All the menus and submenus does the same process as that done in scrambling part, Instead the DESCRAMBLING option descrambles the video where the reverse process of block rotation and block shuffling is done. Finally we recreate the original Video file.

Inaccurate input data is the most common case of errors in data processing. Errors entered by data entry operators can control by input design. Input design is the process of converting user-originated inputs to a computer-based format. Input data are collected and organized into group of similar data.

4.2 PROCEDURE DESIGN

The entire application consists of four modules,

1. Selective Bit Scrambling
2. Block Shuffling
3. Block Rotation
4. Security Analysis

PROCEDURE:

Step1: Open AVI Video.
Step2: Extract Frames into BMP images.
Step3: For each image:
- 3.1: Read Image data
- 3.2 Apply Block Rotation
- 3.3 Apply Line Shuffling
- 3.4 Apply Sign Encryption

Step4: Build AVI video from the scrambled images.

4.3 OUTPUT DESIGN

Initially we select the video file that is to be scrambled in the scrambling interface. Then these are being extracted into JPEG format. This JPEG frames are converted into BMP format. Then in scrambling process the output is the scrambled video file which has undergone block shuffling and blocks rotation of those frames. Then the creation of the video file occurs.

Then in the descrambling interface the same scrambled video file is being selected. Then the extraction is being done here, where the same extraction into JPEG format and converted BMP format frames are being obtained. After this again the reverse process of block shuffling and block rotation is being done. Then the final descrambled video which is otherwise the original video file is the final output of the entire project.

CHAPTER 5
TESTING

5.1 SYSTEM TESTING

5.1.1 UNIT TESTING

1. Test for application window properties.

The Properties of the windows are properly aligned and displayed perfectly.

2. Test for mouse operations.

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

5.1.2 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.

5.1.3 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 6

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 have shown in this project 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 compression techniques that are used enables the compression of the video files that are being used and it reduces the disk space that is being required.

The cryptographic algorithms increase the security of the project. These algorithms are used for encryption and decryption. The security of the project is better than that of the various other related competitors. The simple interface enables the access of various levels of users. The ability to scramble large video files at a good speed is another striking feature. 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.

CHAPTER 7
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 the 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 interface may also be further developed and made more attractive with increased options and Toolbar can also be included in the user interface.

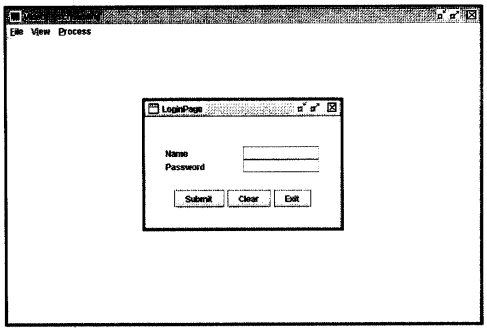
The player can be further enhanced by the inclusion of more controls and the formats that are being supported can also be increased which results in the ability to watch various video formats.

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.

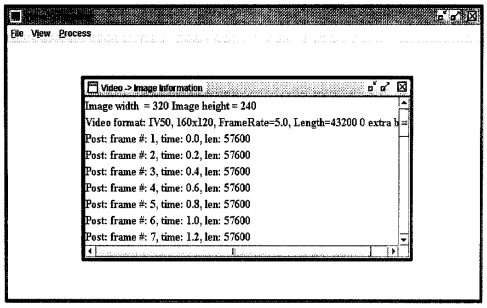
CHAPTER 8

APPENDIX

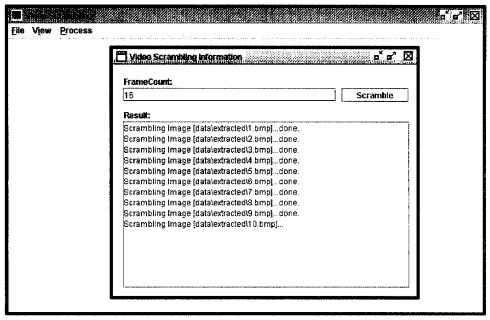
A.1 SAMPLE SCREEN SHOTS



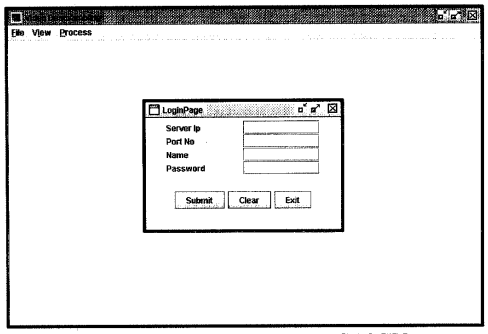
A.1 VIDEO SCRAMBLING LOGIN



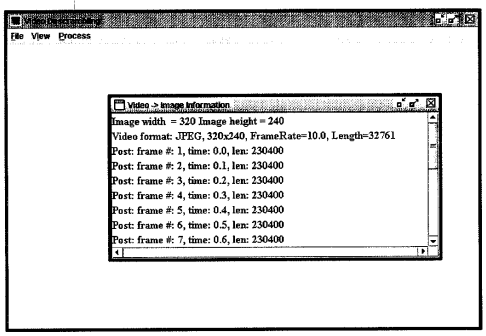
A.2 VIDEO EXTRACTION FOR SCRAMBLING



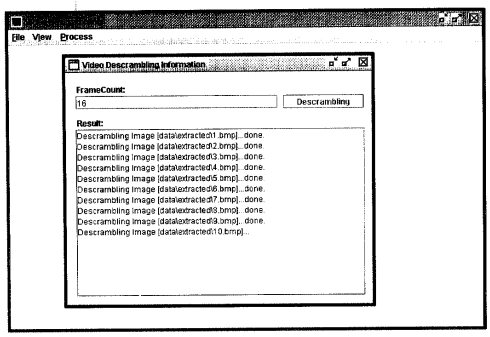
A.3 IMAGE CONVERSION FOR SCRAMBLING



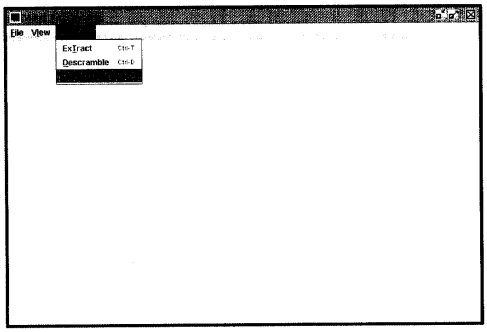
A.4 VIDEO DESCRAMBLING LOGIN



A.5 VIDEO EXTRACTION FOR DESCRAMBLING



A.6 IMAGE CONVERSION FOR DESCRAMBLING



A.7 ORIGINAL VIDEO FILE RECREATION

A.2 SOURCE CODE LIST

```
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.txtaMessage,object,ob
jtextArea);
                        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_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_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.txtaMessage,object,ob
jtextArea);
            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\\","data\\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
```


REFERENCES

PRINT MEDIA

1. Mathew Thomas, **"A tour of Java Swing – Guide"**, PHI, 2000.
2. M C Bridge, **"Image Compression "**, Pearson Education, Edition 2000.
3. Patrick Naughton, **"Complete Reference Java "**, Tata McGraw Hill, Edition 2001.
4. Roger.S.Pressman, **"Software Engineering A Practioner's 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/