Security Mechanisms for Multimedia Networking

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Ali Şaman Tosun, B.Sc, M.S.

* * * * *

The Ohio State University

2003

Dissertation Committee:

Dr. Wu-chi Feng, co-adviser
Dr. Dong Xuan, co-adviser
Dr. Hakan Ferhatosmanoglu
Prof. Junko Shigemitsu

Approved by

Co-Adviser

Co-Adviser

Department of Computer and Information Science
ABSTRACT

With the increased use of multimedia in daily communications, it is necessary to develop efficient and secure transmission mechanisms that are specifically tailored for multimedia. Real-time characteristics and high bandwidth requirements of multimedia data require efficient and scalable mechanisms. For success of commercial multimedia distribution, security mechanisms will be a major factor. Most of the research on multimedia security have focused on watermarking related issues. Security issues related to streaming is not researched in detail and requires further progress.

This dissertation contains my research on security mechanisms for multimedia networking. I have investigated several issues including passive attacks on video streams, secure layered lossless video coding, end-to-end security for proxy-based video distribution and security mechanisms for wireless video distribution. Most the above issues were not researched by the multimedia community and this thesis forms a first step for secure, scalable multimedia networking.

Data transmitted using streaming algorithms depend on available network bandwidth and characteristics of the movie being streamed. Reliance on movie characteristics will reveal information even if the movie is encrypted. Experimental results show that movies can be identified using network traces. To address this problem, I have developed a framework based on MPEG that prevents passive attacks on encrypted movie streams.
Many applications including scientific visualization require lossless video streaming. To facilitate this I have developed a dual-layer framework, an MPEG layer and lossless differences, that provides lossless view of video and reduces bandwidth requirement and security overhead for many settings.

Proxy-based approaches are used for many multimedia applications. I have investigated end-to-end security in proxy based systems with operations for frame dropping, transcoding. My analysis shows that frame dropping can easily be supported using end-to-end security but transcoding is much harder and can be supported in limited form for authentication.

Finally, I have focused on wireless video transmission. For handheld devices with limited capabilities, security mechanisms should be light-weight and introduce minimal overhead. My research on wireless video transmission has shown that encryption overhead can be reduced by using layering and error-preserving encryption.
Dedicated to My Parents
ACKNOWLEDGMENTS

Near the end of my student life, I realize that the years at Ohio State University have been a special and precious time in my life. My experiences here will have a profound impact in my future career. The completion of my study has been made possible through many people's support. I want to take this opportunity to express my sincere appreciation to them.

My adviser, Wu-chi Feng has been a source of inspiration throughout my Ph.D study. His broad knowledge and excellence in teaching in multimedia networking was the motivation of my Ph.D. I want to thank him for introducing me to the field of multimedia networking and providing support during my study.

I am grateful to my dissertation committee members, Dong Xuan and Hakan Ferhatosmanoglu. They have examined my dissertation and provided valuable feedback. Professor Mukesh Singhal and Srinivasan Parthasarathy have also been very helpful thoughtout my study. I would aks also like to thank staff members in CIS department, especially Elley Quinlan, Tom Fletcher, Elizabeth O’Neill.

My parents have been extermely supportive of my academic pursuit. They have always stood behing me in difficuls situations and encouraged me to do better. Withour their support, this dissertation would have never been completed.

I would like to acknowledge former group members Amit Agrawal, Pankaj Sethi and Nagasuresh Seelam for their help. Sencer Kutlug, Murat Demirbas, Atakan
Dogan and many other friends in Columbus have been part of my life. Their abundant help and encouragement can never be overestimated.
VITA

1973 ........................................... Born, Ankara, Turkey

1995 ........................................... B.S. Computer Engineering and
Information Science, Bilkent
University, Ankara, Turkey

1997 ........................................... M.S. Computer Engineering and
Information Science, Bilkent
University, Ankara, Turkey

1998 ........................................... M.S. Computer and Information
Science, The Ohio State University

1997-2003 ................................... Graduate Teaching and Research
Associate, The Ohio State University

PUBLICATIONS

Research Publications

Ali Saman Tosun, Hakan Ferhatosmanoglu “Vulnerabilities in Similarity Search Based
Systems”. \textit{International Conference on Information and Knowledge Management
(CIKM 2002)}, November 2002


Wireless Video Transmission”. \textit{ACM Multimedia Conference (MM 01)}, October 2001

Video Transmission”. \textit{International conference on Information Technology: Coding
and Computing (ITCC 2001)}, April 2001


**FIELDS OF STUDY**

Major Field: Computer and Information Science
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Abstract</strong></td>
<td>ii</td>
</tr>
<tr>
<td></td>
<td><strong>Dedication</strong></td>
<td>iv</td>
</tr>
<tr>
<td></td>
<td><strong>Acknowledgments</strong></td>
<td>v</td>
</tr>
<tr>
<td></td>
<td><strong>Vita</strong></td>
<td>viii</td>
</tr>
<tr>
<td></td>
<td><strong>List of Tables</strong></td>
<td>xii</td>
</tr>
<tr>
<td></td>
<td><strong>List of Figures</strong></td>
<td>xiii</td>
</tr>
<tr>
<td>1.</td>
<td><strong>Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td><strong>Background</strong></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>2.1 <strong>MPEG-1 Video</strong></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>2.2 <strong>Public-Key Cryptosystems and Digital Signatures</strong></td>
<td>14</td>
</tr>
<tr>
<td>3.</td>
<td><strong>Related Work</strong></td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>3.1 <strong>Video Streaming</strong></td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>3.2 <strong>Time Series Data and Similarity Search</strong></td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>3.3 <strong>Video Security Algorithms</strong></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>3.4 <strong>Proxy-based Video Transmission</strong></td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>3.5 <strong>Layered Video Transmission</strong></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>3.6 <strong>Lossless/Loss-bounded Image and Video Compression</strong></td>
<td>23</td>
</tr>
</tbody>
</table>
4. Passive Attacks on Streaming Video and Prevention Mechanisms . . . . 26
   4.1 Network Model .................................................. 28
   4.2 Attack Model ..................................................... 30
   4.3 Time Series Representation and Similarity Search .................. 30
   4.4 Experimental Results ............................................ 31
   4.5 Prevention of Passive Attacks ................................... 36

5. Lossless Video Transmission and Security Mechanism ......................... 41
   5.1 Proposed Method ................................................ 43
   5.2 Experimentation .................................................. 47
       5.2.1 Compression Performance .................................... 53
       5.2.2 Algorithm Speed ............................................ 54
       5.2.3 Effect of MPEG file size on Overall Compression Ratio ...... 56
       5.2.4 Entropy of Differences ...................................... 63
   5.3 Security Mechanism for Layered Lossless Coding ....................... 64

6. Secure Video Transmission Using Proxies ..................................... 68
   6.1 End-to-end Encryption ............................................ 70
       6.1.1 Frame Dropping .............................................. 70
       6.1.2 Quality Adaptation .......................................... 73
       6.1.3 Secure Proxy Caching ....................................... 77
   6.2 End-to-end Authentication ......................................... 79
       6.2.1 Proposed Authentication Scheme ............................. 80
   6.3 Experimental Results on Encryption ................................ 82
       6.3.1 Frame Dropping .............................................. 82
       6.3.2 Encryption/Decryption Speed ................................. 86
   6.4 Experimental Results on Authentication ................................ 88

7. Wireless Video Transmission and Security Mechanisms ........................ 90
   7.1 The Effect of Bit Errors on MPEG Video Quality for Wireless Networks 90
       7.1.1 Playability of Corrupted Video Streams ..................... 92
       7.1.2 Effect of Bit Error on Picture Quality ....................... 93
       7.1.3 2-bit Errors ................................................. 100
   7.2 Light-weight Security Mechanisms for Wireless Video Transmission ...... 100
       7.2.1 Proposed Encryption Algorithms .............................. 102
       7.2.2 Experimental Results ........................................ 104
   7.3 Video Encryption for Transmission over Wireless Networks ........... 106
       7.3.1 Security Issues Relevant to Wireless Video Transmission ... 106
7.3.2 A Theory for Understanding Error Preserving Encryption Functions ........................................ 107
7.3.3 Applications of error preserving encryption functions ......................................................... 112

8. Conclusion ............................................................................................................................. 126

Bibliography ............................................................................................................................ 129
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Layers within MPEG</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Notation used</td>
<td>14</td>
</tr>
<tr>
<td>4.1 Minimum and maximum frame sizes for Crocodile Dundee and E.T</td>
<td>38</td>
</tr>
<tr>
<td>5.1 Overall compression ratio</td>
<td>55</td>
</tr>
<tr>
<td>5.2 Encoding/Decoding performance</td>
<td>55</td>
</tr>
<tr>
<td>6.1 Frame numbers and average frame size for I,P and B frames</td>
<td>83</td>
</tr>
<tr>
<td>6.2 Timings for gpg and PC</td>
<td>87</td>
</tr>
<tr>
<td>7.1 Classification of bits into categories</td>
<td>92</td>
</tr>
<tr>
<td>7.2 Classification of bits into categories</td>
<td>100</td>
</tr>
<tr>
<td>7.3 Timings for gpg and permutation and P-C</td>
<td>117</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Frame types and dependencies in MPEG</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>MPEG’s hierarchy of video sub-units</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>MPEG-1 I-frame compression process</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Block-based motion compensation in MPEG-1</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>Size of minutes of video for E.T.</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>Size of minutes of video for Crocodile Dundee</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>Network model</td>
<td>29</td>
</tr>
<tr>
<td>4.4</td>
<td>Route from client to server</td>
<td>32</td>
</tr>
<tr>
<td>4.5</td>
<td>Average rank of traces belonging to same video</td>
<td>33</td>
</tr>
<tr>
<td>4.6</td>
<td>Minimum rank of traces belonging to same video</td>
<td>34</td>
</tr>
<tr>
<td>4.7</td>
<td>Maximum rank of traces belonging to same video</td>
<td>35</td>
</tr>
<tr>
<td>4.8</td>
<td>Overhead of secstream</td>
<td>39</td>
</tr>
<tr>
<td>5.1</td>
<td>Compression and decompression using proposed method</td>
<td>43</td>
</tr>
<tr>
<td>5.2</td>
<td>Snapshot of system based on MPEG-1</td>
<td>45</td>
</tr>
<tr>
<td>5.3</td>
<td>Sample head image</td>
<td>47</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.4</td>
<td>Sample wind image</td>
<td>48</td>
</tr>
<tr>
<td>5.5</td>
<td>Compression ratio of frames for head data for lossless compression</td>
<td>50</td>
</tr>
<tr>
<td>5.6</td>
<td>Compression ratio of frames for wind data for lossless compression</td>
<td>50</td>
</tr>
<tr>
<td>5.7</td>
<td>Compression ratio of frames for head data for loss-bounded compression with bound 2</td>
<td>51</td>
</tr>
<tr>
<td>5.8</td>
<td>Compression ratio of frames for wind data for loss-bounded compression with bound 2</td>
<td>51</td>
</tr>
<tr>
<td>5.9</td>
<td>Compression ratio of frames for head data for loss-bounded compression with bound 4</td>
<td>52</td>
</tr>
<tr>
<td>5.10</td>
<td>Compression ratio of frames for wind data for loss-bounded compression with bound 4</td>
<td>52</td>
</tr>
<tr>
<td>5.11</td>
<td>SNR of wind dataset</td>
<td>56</td>
</tr>
<tr>
<td>5.12</td>
<td>Compression ratio of wind dataset</td>
<td>57</td>
</tr>
<tr>
<td>5.13</td>
<td>SNR of head dataset</td>
<td>57</td>
</tr>
<tr>
<td>5.14</td>
<td>Compression ratio of head dataset</td>
<td>58</td>
</tr>
<tr>
<td>5.15</td>
<td>Single layer lossless video compression vs our scheme on head data</td>
<td>58</td>
</tr>
<tr>
<td>5.16</td>
<td>Single layer lossless video compression vs our scheme on wind data</td>
<td>59</td>
</tr>
<tr>
<td>5.17</td>
<td>Entropy of differences for head images with lossless compression</td>
<td>60</td>
</tr>
<tr>
<td>5.18</td>
<td>Entropy of differences for wind images with lossless compression</td>
<td>61</td>
</tr>
<tr>
<td>5.19</td>
<td>Entropy of differences for head images with lossbound 2 compression</td>
<td>61</td>
</tr>
<tr>
<td>5.20</td>
<td>Entropy of differences for wind images with lossbound 2 compression</td>
<td>62</td>
</tr>
<tr>
<td>5.21</td>
<td>Entropy of differences for head images with lossbound 4 compression</td>
<td>62</td>
</tr>
<tr>
<td>5.22</td>
<td>Entropy of differences for wind images with lossbound 4 compression</td>
<td>63</td>
</tr>
</tbody>
</table>
5.23 Lossless differences for sample wind image .................................. 65
5.24 Canny edge detection on lossless differences ................................. 65
5.25 Canny edge detection on original image ....................................... 66
6.1 Network model for proxy-based systems ....................................... 69
6.2 Frame dropping in encrypted stream .............................................. 71
6.3 Original frame ................................................................. 75
6.4 All DCT coefficients flipped to positive ........................................ 76
6.5 Block diagram of end-to-end authentication ................................. 80
6.6 Percentage of data that can be dropped in encrypted form in level 1 . 84
6.7 Percentage of data that can be dropped in encrypted form in level 2 . 85
6.8 Percentage increase in size due to fillers .................................... 86
6.9 Overhead as a function of quality Levels for Crocodile Dundee ....... 87
6.10 Overhead as a function of quality levels for E.T ............................. 88
7.1 I frame used in the experiment ............................................... 91
7.2 Grayscale image ............................................................... 94
7.3 Errors in 3 types of frames .................................................... 95
7.4 Errors in I frame on other frames ............................................. 96
7.5 Errors in P frame ............................................................... 97
7.6 Macroblock size for I, P and B frames, white means large macroblock, black means small macroblock ............................. 98
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.7</td>
<td>Errors for I, P and B frames, white means fewer errors, black means more errors.</td>
<td>99</td>
</tr>
<tr>
<td>7.8</td>
<td>Percentage of undetected bit-errors for I, P and B frames, white means high percentage, black means small percentage.</td>
<td>118</td>
</tr>
<tr>
<td>7.9</td>
<td>Average PSNR value for different breakpoints</td>
<td>119</td>
</tr>
<tr>
<td>7.10</td>
<td>Layered video encryption</td>
<td>119</td>
</tr>
<tr>
<td>7.11</td>
<td>VEA encryption</td>
<td>120</td>
</tr>
<tr>
<td>7.12</td>
<td>Extended VEA encryption</td>
<td>120</td>
</tr>
<tr>
<td>7.13</td>
<td>BCH code vs the number of frames in error for various error rate.</td>
<td>121</td>
</tr>
<tr>
<td>7.14</td>
<td>BCH code vs percentage of overhead</td>
<td>121</td>
</tr>
<tr>
<td>7.15</td>
<td>Encryption for wireless video transmission</td>
<td>121</td>
</tr>
<tr>
<td>7.16</td>
<td>Example of an error preserving function</td>
<td>122</td>
</tr>
<tr>
<td>7.17</td>
<td>Example of a function which is not error preserving</td>
<td>122</td>
</tr>
<tr>
<td>7.18</td>
<td>Sample isometries of square</td>
<td>123</td>
</tr>
<tr>
<td>7.19</td>
<td>Byte distribution for flower.mpg</td>
<td>123</td>
</tr>
<tr>
<td>7.20</td>
<td>Byte distribution for bus.mpg</td>
<td>124</td>
</tr>
<tr>
<td>7.21</td>
<td>Double byte distribution for flower.mpg</td>
<td>124</td>
</tr>
<tr>
<td>7.22</td>
<td>Double byte distribution for bus.mpg</td>
<td>125</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

Over the last couple of years there has been many improvements in networking technologies. Today 75% of US homes have the infrastructure for broadband Internet connections including broadband cable and DSL. Multimedia data requires high bandwidth therefore broadband connections provide the basic building blocks for multimedia communication over the Internet. Currently around 10% of US homes have broadband Internet connections. Although the infrastructure is ready many users do not sign up for broadband connections, which cost $40 - $50 per month ($20 more expensive per month than narrowband connections). The missing link is the lack of compelling applications which can entice users to broadband connections. Success of commercial applications will depend upon availability of digital rights management solutions for multimedia data.

Multimedia has emerged as a common form of information. Multimedia content is currently available on a large number of sites on the Internet. However, most of the video data is in the form of short video clips. I expect to see video content in the form of long movies on the web in the near future. Multimedia applications are CPU-intensive and require high-bandwidth therefore it is necessary to develop efficient transmission strategies for multimedia. Streaming video is such an example. A wide
variety of multimedia applications such as lossless video and video-on-demand will be available in the future. Most of the research so far has focused on video-on-demand (including areas such as server design) and video streaming. Efficient strategies are needed for applications based on lossless video, particularly for scientific computing activities.

As multimedia data becomes more prominent in our day to day lives, the need for efficient security mechanisms will become increasingly more important. Examples of such future applications include video e-mail and video-on-demand. In particular, as more commercial sites offer video and audio-based services, efficient and secure encryption of video data will probably determine the acceptance or failure of these applications. Security mechanisms for video data consists of encryption, authentication schemes and watermarking schemes. Encryption is used to make sure that no unintended person can get the original video content during transmission. Authentication is used to verify that video content comes from the person who claims to be sending it. A watermark inserted into the movie indicates the user and prevent users from distributing it to others.

While encryption methods to secure video streams exist, they can consume significant processor resources in addition to the processing requirements of the video stream itself. As we move to higher quality bit streams, such as MPEG-2, the need for a light-weight encryption method is even more important since the decompression itself will require nearly all of the processor’s resources. When proxies are used to adapt video streams, encryption causes additional problems since proxy has to decrypt the data, perform proxy operation and encrypt the data to send it to the client.
Proxies typically handle multiple clients so using encryptions more computational resources are needed at the proxy. Light-weight encryption will alleviate this problem since less computational resources will be used for encryption.

Wireless networks emerged as a hot research area over the last couple of years. Commercially available products based on wireless networks such as PDAs and cellular phones have reached a large consumer base. Users’ demand in wireless Internet connection to read email and browse the web culminated in commercially available wireless Internet connection. Wireless multimedia applications such as online games or videos is expected to become feasible in the near future. Cellular phones with digital cameras is a promising indicator of this. Demand for wireless video transmission will encourage development of efficient strategies. Due to limited processing power of wireless devices and limited bandwidth available in wireless networks, video transmission over wireless networks is a challenging problem. Several video coding algorithms such as H.263, MPEG4 have been developed for wireless video transmission. 3G wireless networks are a huge step towards practical wireless video. Initially 3G will offer 144 kbps bandwidth and this will go up over time. Such bandwidth is enough to support wireless video transmission.

When we mix digital rights management and the transmission of the data over wireless networks, several bad side-effects manifest themselves when transmitted over wireless networks. First, many of the standard encryption methodologies that exist today require that all bits received after the transmission of data are correct. This is due mainly to some of the inherent properties required in designing good cryptographic systems. As a result of this, one can expect significant overhead due to
retransmissions and forward-error-correction in applying standard encryption techniques to multimedia data. Second, by requiring that all the bits received correctly, the systems cannot take advantage of the fact that some of the multimedia applications are somewhat tolerable to network bit errors.

Many multimedia applications deployed over the Internet use proxies to handle unpredictable bandwidth, heterogeneity of receivers and caching. For MPEG coded video proxy operations are frame dropping and quality adaptation. In frame dropping some of the frames are dropped from the coded stream resulting in a lower frame rate. In quality adaptation, video stream is partially decoded to alter quantization and recoded to get smaller size. When encryption is used, one natural question that arises is whether it is possible to perform proxy operations on encrypted data instead of decrypting the data and performing the operations. Proxy operations on encrypted data will be more efficient and is preferred for building scalable system. One of the major goals of designing secure transmission schemes for video is to add as few complexity to the proxies as possible. Proxies will handle multiple clients and combined computational overhead may require additional hardware at the proxies.

Many scientific applications including visualization and medical imaging desire lossless video transmission. Lossless video transmission guarantees that all the details in the image is kept during compression. The disadvantage of lossless video coding is that the compression ratio that can be achieved is very low in the range of 2 to 3 depending on the video. MPEG compression which is lossy can yield compression ratios of 10 to 20 and even higher depending on quality of video desired. Efficient
methods are needed to combine these 2 methods with the goal of getting better compression ratio and providing a lossless image to user. Efficient security mechanisms are needed for such a system.

Secure streaming is vital for commercial streaming of stored videos but is not investigated in detail. Whether encryption of transmitted data provides us secure streaming needs to be addressed. Stored video, real-time requirements, use of small buffers for streaming can render encrypted video vulnerable. The amount of data sent using streaming video protocols depends on available network bandwidth and characteristics of the video being streamed. Even if video is encrypted information regarding the stream is revealed by the transmission schedule. Whether this information can be used to identify the movie needs to be investigated and if movies can be identified prevention mechanisms are needed to preserve privacy of the user. Protecting privacy of user is important for commercial secure streaming applications.

My research focuses on the above mentioned issues which can be grouped as efficient and secure transmission of multimedia over both wired and wireless networks in both end-to-end systems and proxy based systems. My goal is to design schemes to achieve both efficiency and security goals.

Rest of the thesis is organized as follows: In the next chapter background on MPEG video compression standard and digital signatures is provided. Chapter 3 describes related work on video streaming, time series data, similarity search, Video security algorithms, proxy-based video transmission and layered video transmission. Passive attacks on streaming video and prevention mechanisms is discussed on chapter 4. Lossless layered video coding and associated security mechanism is described in chapter 5. Chapter 6 includes end-to-end security mechanisms for proxy-based
video streaming. Wireless video transmission including effect of bit errors, layered video transmission and video encryption algorithms for wireless video is described in chapter 7. Finally, I conclude with chapter 8.
CHAPTER 2

BACKGROUND

Digital video technology has evolved tremendously in compression and several video compression formats including MPEG-1, MPEG-2, H.262, QuickTime, RealMedia and Windows Media. In this chapter, I will briefly describe the MPEG video compression format to give the readers an overview of video compression. The MPEG compression format discussed here will render the coming chapters easier to understand. In addition, I will provide a brief summary of public-key cryptosystems and digital signatures.

2.1 MPEG-1 Video

The MPEG video compression standard is a layered, DCT-based video compression standard that results in VHS quality compressed video stream that has a bit rate of approximately 1.5 Mbits/second at a resolution of approximately 352x240 pixels. At a high level, MPEG video sequences consist of several different layers that provide the ability to randomly access a video sequence as well as provide a barrier against corrupted information. The layers within MPEG are shown in table 2.1.

All MPEG frames are encoded in one of three different ways: Intra-coded (I-frames), Predictive-coded (P-frames), or Bidirectionally-predictive-coded (B-frames).
<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence layer</td>
<td>random access unit: context</td>
</tr>
<tr>
<td>Group of pictures layer</td>
<td>random access unit: video</td>
</tr>
<tr>
<td>Picture layer</td>
<td>Primary coding unit</td>
</tr>
<tr>
<td>Slice layer</td>
<td>Resynchronization unit (within picture)</td>
</tr>
<tr>
<td>Macroblock layer</td>
<td>Motion compensation unit within slice</td>
</tr>
<tr>
<td>Block layer</td>
<td>DCT Unit within macroblock</td>
</tr>
</tbody>
</table>

Table 2.1: Layers within MPEG

![Frame types and dependencies in MPEG](image)

Figure 2.1: Frame types and dependencies in MPEG

I-frames are encoded as discrete frames, independent of adjacent frames. Thus, they provide randomly accessible points within the video stream. Because of this, I-frames have the worst compression ratio of the three frame types. P-frames are coded with respect to a past I-frame or P-frame, resulting in a smaller encoded frame size than the I-frames. The B-frames require a preceding and a future frame, which may be either I-frames or P-frames, in order to be decoded, but they offer the highest degree of compression. For GOP structure IBBPBBP frame types and dependencies between frames is given in Figure refframes. The dependency between frames is shown with an
The MPEG video at the highest level is called a sequence. The sequence consists of a collection of GOPs (Group of Pictures) which in turn is a group of frames. Each frame in a GOP has a type and encoded accordingly. In general any type of…
frame is divided into \textit{slices}, that run horizontally and could wrap around the frame edge. Each slice encloses a group of macroblocks. This hierarchical structure of MPEG is shown in Figure 2.2 Each sequence/GOP/frame/slice/macroblock unit has a start code and a header at the beginning and an ending marker code to indicate the respective boundaries. This hierarchical division of video allows error tolerance and error recovery possible by synchronizing the decoder to a start code. In the event of an error, an MPEG player can search the stream for a byte-aligned marker that indicates either the sequence, group of pictures, picture, or slice layer.

MPEG-1 is a Discrete Cosine Transform (DCT) based compression standard. DCT transformation is performed on a $8 \times 8$ block. Figure 2.3 shows the compression process of MPEG-1 video encoding standard.

First 16 macroblocks in RGB (red, green, blue) space are converted to YUV space using the linear transformation given below.

\[
\begin{pmatrix}
Y \\
U \\
V
\end{pmatrix} =
\begin{pmatrix}
0.299 & 0.587 & 0.114 \\
-0.147 & -0.289 & 0.436 \\
0.615 & -0.515 & -0.100
\end{pmatrix}
\begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]  

(2.1)

Then macroblocks in the Y channel are then divided horizontally and vertically to obtain four $8 \times 8$ blocks. The U and V channels are sub-sampled to obtain one $8 \times 8$ block per channel ore two $8 \times 8$ blocks per channel. The above results in a total of six $8 \times 8$ blocks - four Y blocks, one U block, one V block. Each $8 \times 8$ block is then run through discrete cosine transform given by the equation

\[
F(u, v) = \frac{1}{4} C(u)C(v) \sum_{i=0}^{7} \sum_{j=0}^{7} F(i, j) \cos \frac{(2i + 1)u\pi}{16} \cos \frac{(2j + 1)v\pi}{16}
\]  

(2.2)
Figure 2.3: MPEG-1 I-frame compression process.
where $F(u,v)$ are transform coefficients, $f(i,j)$ are pixel values and $C(x)$ is defined as

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases}$$

(2.3)

This results in $8 \times 8$ block of 64 DCT-coefficients. These blocks are called “DCT-blocks”. The DCT-coefficients are then quantized and rounded off to reduce their size in order to improve the prospects of coding these numbers using fewer bits thereby achieving compression. At this time, an important decision is made regarding the quantized coefficients. By throwing away higher order coefficients in the zig-zig ordering of the numbers of the block, better compression is achieved without any significant loss in visible quality to the user. So, numbers above a certain frequency zone are made zero when their size is below a predetermined threshold.

In the next stage, the numbers of the block are run through entropy encoding which is basically variable length huffman run-length encoding process. These blocks are then demarcated using end-of-block marker bits. In effect each $16 \times 16$ pixel macroblock results in 6 compressed blocks.

As mentioned earlier there are three types of frames in MPEG-1. The $I$ type frame is an intra coded frame. A $P$ or $B$ frame is different where other frames are used as references when they are coded. Therefore they cannot be decoded unless the reference frames are available and are first decoded. For the most part, compression of $P$ and $B$ frames is not different from that of $I$ frame. The only difference is the use of block based motion vectors in these frames(refer to Figure 2.4). Before encoding a $P$ frame’s macroblock a process called motion vector search is performed wherein a closely matching macroblock from a past $I$ and/or $P$ frame is first found out. Instead
of coding the original macroblock, the differences with this matching macroblock are encoded, along with a displacement vector of the matching block with respect to the original block. MPEG does not specify how to do motion compensation but the key idea is that in a moving picture sequence, two adjacent frames in time will have most of the image parts, the same, except for their location in the image and small fluctuations in the color intensity due to natural disturbances and variations in the conditions of the environment.

The frame structure and the lossy nature of MPEG compression allows for scaling the bit-rate of an already encoded video by means of re-compression. The process involves one or all of the techniques like re-quantization, GOP pattern redesigning, temporal scaling such as frame dropping, spatial scaling such as resolution reduction and discarding of DCT coefficients. In re-quantization, the original blocks are reconstructed in the DCT-domain and are re-quantized to a coarser granularity. This results in more compression at the cost of further loss of information. Frame-dropping is performed to reduce the frame-rate of the video. In this process some un referenced
<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_s$</td>
<td>private key of server</td>
</tr>
<tr>
<td>$d_s$</td>
<td>public key of server</td>
</tr>
<tr>
<td>$e_c$</td>
<td>private key of client</td>
</tr>
<tr>
<td>$d_c$</td>
<td>public key of client</td>
</tr>
<tr>
<td>$h$</td>
<td>cryptographic hash function</td>
</tr>
</tbody>
</table>

Table 2.2: Notation used

frames are discarded and the receiver skips their display. Spatial scaling techniques such as resolution reduction involves re-encoding the video with smaller frame dimensions. Discarding DCT-coefficients results in compression gains by needing to code fewer numbers and achieving tighter run length encoding.

2.2 Public-Key Cryptosystems and Digital Signatures

In a public-key cryptosystem each user has 2 keys: a private key and a public key. It is computationally infeasible to determine private key from the public key. So the public key can be revealed to all parties and private key is kept secret. A message encrypted using private key can be decrypted by the public key and a message encrypted using public key can be decrypted using the private key. Consider a communication between a client and a server. Using the notation given in Table 2.2, if client wants to send a message $m$ to server, client encrypts with public key of server and sends $d_c(m)$ to server. Using private key server can recover message $m$.

Digital signature is a method for signing a message in electronic form. Using a public key cryptosystem such as RSA, encryption with the private key of sender can be used as a signature scheme. The server sends $d_s(x)$ to the client for message $x$.  

14
The receiver can then verify the authenticity by computing $e_s(d_s(x))$. Since $e_s$ is the public key, the client knows $e_s$. Above signature scheme called RSA signature has high computational cost. It is possible to reduce computational overhead of digital signatures using cryptographic hash functions. A short (typically around 20 bytes) message digest is computed using cryptographic hash function and this message digest is signed. The pair consisting of original message and signed message digest are sent to client $(x, d_s(h(x)))$. The client computes $h(x)$ for the message, decrypts the signed message digest and compares them. If they are equal then the message is from the server.

Cryptographic hash functions used in digital signatures have the following properties. A detailed formal description is given in [63].

- **Preimage Resistant**: Given a hash value $y$, it is computationally difficult to find an $x$ such that $h(x) = y$. The interpretation of this is given a hash value we can not find a message with that hash value.

- **Second Preimage Resistant**: Given $x$, it is computationally difficult to find $x' \neq x$ such that $h(x') = h(x)$. Given a message it is difficult to find another message with the same hash value. This property is crucial for digital signatures. Without this property an attacker can insert $x'$ instead of $x$, keeping hash value unchanged and users will think this is authentic message.

- **Collision**: It is computationally difficult to find $x \neq x'$ such that $h(x') = h(x)$. This is important since cryptographic hash functions compute a fixed length hash for arbitrary length messages.
Many cryptographic hash functions have been proposed in the literature. A survey of cryptographic hash functions is given in [8]. The following structure is suggested for designing hash functions: First, the message is padded to be a multiple of $t$ where $t$ is the size of segment on which operations are performed. The padded message is divided into $t$-bit blocks. Round function is performed on the blocks using the chaining method. In the chaining method, the previous result and next unused block are fed into the round function to get a new result. The final result is the hash value. An initial vector is used together with the first block to get initial result. To have a strong one-way hash function, the round function should be chosen as a good one-way function. A typical hash length is 16-20 bytes.
CHAPTER 3

RELATED WORK

In this chapter, I describe the related work necessary to understand the remainder of the thesis. The related work is divided into subsections along the lines of the chapters.

3.1 Video Streaming

Streaming video has received a lot of interest with the increase of multimedia data on the Internet. Several approaches have been proposed for the integrated delivery of streamed video across the Internet from both industry and academia. On the industry side Microsoft’s Windows Media Player, RealNetwork’s RealAudio/RealVideo and Quicktime from Apple all use their own proprietary streaming formats. Systems developed by academia include VIC - a video conferencing tool built for the MBone [47], the Continuous Media Toolkit from the University of California - Berkeley [60], the Quasar Adaptable Streaming Video Player from the Oregon Graduate Institute [69], the Vosaic streaming video protocol from the University of Illinois [18]. Most of these systems use reliable transmission for control information while using unreliable transmission for the delivery of video streams. Some of them are more TCP compatible than others. More recently, streaming media from multiple sources was proposed
PeerCast [22] streams live media using an overlay tree formed by clients and CoopNet [53] proposes mechanisms for cooperation of clients to distribute streaming video when server is overloaded. A peer-to-peer media streaming model is proposed in [76] and a hybrid CDN, peer-to-peer media distribution aimed at efficient use of CDN resources is given in [75]. A somewhat related paper to our passive attack approach is [65] which investigates the identifiability of World Wide Web traffic based on unconcealed information.

3.2 Time Series Data and Similarity Search

Streaming traces can be represented as time series data and similarity searches can be used as a tool to identify the movie. In this section we give a brief overview of work on time series data that can be used for scalable, efficient attacks.

Time series data such as stock quotes are common in database applications and efficient methods have been developed to answer queries on time series data. The most common query in time series data is a similarity search where the time series representation of a stock quote is given and stocks that are similar are requested. Time series data is usually viewed as a point in a high dimensional space and similarity search between the two time series data is computed by using a distance in high dimensional space [2, 25, 36]. The Euclidean distance defined by \( \| x - y \| = (\sum (x_i - y_i)^2)^{1/2} \) where \( x \) and \( y \) are n-dimensional points (time series data with n attributes) is the most common distance metric used. The main advantage of high dimension representation of time series data is that multi-dimensional access methods [30, 10] can be used for similarity search. For long time series data (very high dimension), multi-dimension access methods are inefficient and dimensionality reduction techniques are
used. The main idea in dimensionality reduction is to transform the coefficients using a Discrete Fourier Transform (DFT) or Discrete Wavelet Transform (DWT) and approximating the time series data with a few coefficients [2, 25, 42]. For queries of the form “Find me all sequences whose distance from \( x \) is less than \( k \),” dimensionality reduction techniques guarantee that every time series data with distance less than \( k \) will be retrieved, although some additional sequences can also be retrieved and a post-processing step is needed to remove them. Besides euclidean norm \( L_\infty \) and \( l_p \) norms are also used for time series data. A similarity search scheme that takes into account presence of noise, scaling, and translation is proposed in [3] and efficient time series indexing is extended to \( l_p \) norms in [78]

Similarity search have emerged as a common form of query in modern database applications such as multimedia information systems [64, 24], CAD/CAM [12], geographical information systems (GIS), medical imaging [38, 37], time-series databases and bioinformatics. The data is usually represented by a feature vector which summarizes the original data with some number of dimensions. The similarity between two objects is defined with a distance function, e.g., Euclidean distance, between the corresponding feature vectors. For example, in image databases, the user may pose a query asking for the most similar images to a given image [5]. 3D Shape histograms are used in molecular biology to find similar 3D proteins [4]. Similarity query with multi-dimensional data is usually implemented by finding the closest feature vector(s) to the feature vector of the query data. This type of query is known as a nearest neighbor (NN) query [59] and it has been extensively studied in the past [33, 72, 6, 11, 13, 19].
3.3 Video Security Algorithms

Historically, the straight-forward application of standard encryption algorithms to entire video streams adds too much computational overhead because all the data needs to be encrypted and decrypted and is typically very large. Because MPEG decoding is a computationally intensive process, solutions have been proposed to add minimal complexity for encryption and decryption.

The ZigZag-Permutation algorithm, which permutes the AC coefficients, is one of the earliest algorithms designed for MPEG encryption [41]. In this technique, the DCT coefficients are encoded in a different order than the standard MPEG video zig-zag order. Thus, in order to decrypt the sequence, an attacker must know the order in which the coefficients are encoded. By reordering the coefficients as they appear in the compressed bit-stream, this algorithm results in slightly lower compressions ratios due to the decrease in the average run-length size. This algorithm is very fast but has been shown to have serious security problems [40]. The main problem is the fact that only the AC values are permuted leaving all header information in plaintext, allowing for simple guessing algorithms to break the encryption.

In [66], the authors propose a real-time, software encryption algorithm using selective coding of I-frames. This approach encrypts the headers and the I-frames using the DES encryption algorithm. It has been shown that this technique has security problems due to the intracoded macroblocks present in unencrypted P and B frames [32]. Intracoded macroblocks are encoded independently of other frames and blocks and thus their presence in P and B frames can be decoded correctly without the I-frame. So even if only I-frames are encrypted, we can see parts of the image in P and B frames because of these I-blocks.
Qiao and Nahrstedt proposed a secure MPEG encryption algorithm which uses DES [39]. It encodes only half of the data in a stream using DES and has a 47 percent reduction in the number of XOR operations executed over DES. This approach can be performed in real-time.

Shi and Bhargava proposed a scheme which flips the signs of the DCT coefficients based on a key [16]. This algorithms also meets the real time requirements of MPEG video applications.

The Permutation-Complementation algorithm [20] is a light-weight video encryption algorithm designed for wireless video transmission. It preserves the number of bit errors that may be left in data after FEC decoding. Thus, it is amenable for wireless video transmission.

Most of the research in authentication of multimedia is on image authentication. Especially image watermarking [48, 44] has received a lot of attention over the last few years. Image watermarks embeds data into an image and embedded data can be used for authentication. Video authentication have received limited interest so far. Issues in MPEG video authentication is investigated and several solutions are proposed in [43]. Compression-tolerant features of image/video are extracted and signed using a digital signature in [68].

3.4 Proxy-based Video Transmission

Proxy-based dissemination architectures focus on actively managing video content through a proxy. In proxy prefix caching [62], the proxy stores the initial frames of popular clips. When the proxy receives a request, it initiates transmission to the client
and requests the remaining frames from the server. This hides delay, throughput and loss effects of poor transmission conditions between the proxy and server.

Video Staging [79] aims to reduce the backbone WAN bandwidth requirement by using available storage space at proxy servers. Only part of the video stream is retrieved from the video server across the backbone WAN, while the rest of the video stream is delivered to users locally from proxy servers attached to LANs.

A proxy caching mechanism for layered-encoded multimedia streams is proposed in [57]. It provides a prefetching mechanism to support higher quality cached streams during subsequent playbacks and improve the quality of cached stream with its popularity. A fine-grained replacement algorithm suited for layered-encoded streams is also proposed.

An overview of the challenges in end-to-end security in proxy-based systems is provided in [56]. To our knowledge there are no papers that provide secure video transmission schemes using proxies.

3.5 Layered Video Transmission

Layered video transmission has been proposed for a number of purposes. The main advantage of layered techniques is that it allows different resource management policies to be applied to the various layers independently. Several examples are given in the rest of this subsection.

If a network supports differentiation among important and less important data (such as the cell loss priority bit in ATM), layered coding can be used to help protect the more important parts of the video. Examples for MPEG video include

• giving higher priority to I-frames in MPEG [15],
• giving higher priority to the lower numbered AC coefficients in DCT-based video over wired and wireless networks [23],

• giving higher priority to motion vector data for error recovery [46] , and

• giving higher priority to additional information that aids in the recovery of lost data (including slice recovery information) [52].

Performance of prioritization schemes in terms of traffic characteristics, SNR, and visual quality has been investigated in [7] and a bandwidth allocation scheme has been proposed.

Layered coding can be used as the underlying mechanism of secure video transmission using proxies. Each layer can be encrypted independently and a proxy can send as many layers as bandwidth permits to the client. Depending on the number of layers, some layers need not be encrypted [67]. The main problem with layered coding using MPEG is that header information (Picture header, Slice header etc) are typically replicated and can cause high overhead depending on the number of layers. It also does not support fine-grained control of adaptation.

3.6 Lossless/Loss-bounded Image and Video Compression

There has been considerable research in the area of lossless image compression in the last three decades. While many standard compression techniques such as Huffman, Arithmetic, Lempel-Ziv (LZ), and Lempel-Ziv-Welch (LZW) can be employed, lossless image compression techniques attempt to take advantage of the spatial properties of the image to aid in compression. A number of image-specific lossless compression algorithms and systems have been introduced recently. These algorithms
include the JPEG lossless coder, SUNSET, universal context modeling, FELICS [31], CALIC [77], LOCO-I, SICLIC and many others. Despite all the effort, there has not been much improvement in the compression ratio which typically varies from 1.5:1 to 3:1, depending on the image. Even with such low compression ratios, the gains can have substantial impact, given the size of the uncompressed original image. Lossless image compression algorithms typically comprise of two distinct and independent components: modeling and coding. The modeling part can be formulated as an inductive inference problem in which an image is observed pixel by pixel in some defined order (usually in raster-scan). The goal then is to infer the next sample value from a selected region of pixels that are close to it. For this paper, we focus on the use of the LOCO-I image compression algorithm since it has been standardized as the algorithm for JPEG 2000 [73]. LOCO-I provides for both lossless and near-lossless compression of continuous-tone images. Since LOCO-I has been designed for still-image data, it does not take into account spectral redundancy in the case of multi-spectral images and temporal redundancy in the case of video sequences. It is based on a simple fixed context model - combining simplicity with the compression potential of context models. Based on the context for the current pixel the current pixel value is predicted using a primitive edge detector (because of complexity constraints). The context for conditioning the current prediction residual is built out of the differences between the pixel values in the context. These differences represent the local gradient thus capturing the local activity. The model is tuned for efficient performance in conjunction with an extended family of Golomb-type codes, which are adaptively chosen and and embedded alphabet extension for coding of low-entropy image regions. Interband CALIC [74] is one step to take into account the above two
resulting in modest gains in compression. Another approach is SICLIC [9] which is an inter-color coding algorithm similar to the LOCO-I algorithm. It does both inter and intra color encoding taking into account the spectral redundancy component in the image.

Lossless video coding is useful in applications where no loss of information or visual quality is tolerable. Although there is a lot of literature on lossless image coding, to our knowledge there are a few papers on lossless video coding. Extension of 2-D prediction based method of lossless image coding to 3-D by treating frames in groups in the temporal domain is investigated in [49]. An embedded to lossless coding is proposed in [1]. Embedded to lossless coding allows decoding the video stream into any bit rate up to lossless bit rate. In [51] a lossless video coding technique based on modification of DCT coefficients with a lossless quantization process is described. To achieve higher compression ratios diagnostically lossless video coding gained interest in the medical imaging community. Application of this to angiogram data using a wavelet-based model is proposed in [29]. These approaches are not designed for transmission over a network, whereas our approach is designed as network transmission as the main goal.
CHAPTER 4

PASSIVE ATTACKS ON STREAMING VIDEO AND PREVENTION MECHANISMS

Streaming video is a common form of information on the Internet today. With the availability of broadband networking in homes, video streaming for entertainment is becoming a reality. Secure streaming will be an important technology for companies providing streaming video service. Companies that provide software for streaming video will most likely include encryption/decryption capabilities and present it as a secure streaming solution.

Information regarding an encrypted video stream can be revealed if encryption is blindly used as a solution to secure streaming. The amount of information transmitted, the length of the streaming session, the packet sizes and the time gap between packets can reveal information that might lead to the discovery of movie being streamed. Typically, a scene which has more action will require higher bandwidth and more information needs to be sent to the client. Parts of the movie with high bandwidth requirement is movie dependent. Some movies will require high bandwidth at the beginning, some at the end and some in the middle. Also, there might be multiple scenes with high bandwidth requirement.
Figure 4.1: Size of minutes of video for E.T.

The size of minute-long movie segments for E.T and Crocodile Dundee are given in Figures 4.1 and 4.2 respectively. As can be seen from the figures there are many differences between the graphs. The initial bandwidth requirement of Crocodile Dundee is about 2 MB per minute whereas that of E.T is less than 1 MB per minute. Crocodile Dundee has a dip in the 63rd minute and E.T has small peak at the 63rd minute. These differences will reveal information even if the video is encrypted because streaming algorithms will send more data when there is a peak and send less data when there is a dip. From a streaming point of view, streaming algorithms will reveal movie characteristics, the question is whether the changes in network conditions render this information useless or not.

There are a couple of interesting questions that arise based on above discussion. Streaming algorithms may reveal information but can the revealed information used
Figure 4.2: Size of minutes of video for Crocodile Dundee

in some way to identify the movies? How difficult is it to launch such an attack? How much resources are needed for such an attack in terms of processing power and space? Is it possible to stream video without revealing any information? In the remainder of this chapter we investigate these issues and try to answer above questions.

4.1 Network Model

We consider video streamed from the same server. Our network model is shown in figure 4.3 There will be multiple routers on the path from the server to the client and these routers have network layer information like the number of packets and the size of packets for a streaming session. A streaming session is identified by the server and client, which is available in the TCP/UDP header.
There are a number of ways of collecting information about stream statistics which can lead to the identification of video being streamed.

- Local area networks such as Ethernet use a broadcast medium and information about the packets such as the server identity, the sizes of packets etc. are available to all nodes on the local area network.

- In wireless networks, packets can be received by all the nodes in the coverage area. Even if encryption is used, information is revealed by packet sizes, the server IP address etc.

- Routers can collect information about the network characteristics. Cisco Netflow\(^1\) services is an example. Netflow services are designed for applications such as network monitoring, network planning and analysis, application monitoring and profiling, user monitoring and profiling. IP addresses, packet and

\(^1\)http://www.cisco.com/go/netflow
byte counts, timestamps, type-of-service, application ports are part of flow data available to Netflow services.

4.2 Attack Model

Passive attacks are aimed at identifying the movie using traces even if the movie is encrypted. Information that can be used for passive attacks include the amount of data transferred, packet sizes, time gap between packets etc. The attack works as follows: A user streams video from the server and stores a trace of stream consisting of packet sizes. This user gets one of the routers on the path from the server to clients and collects information about packet traces. The user then can figure out which other users streamed the same movie by comparing the trace it stored with the traces collected by the router.

Although we define this as an attack, it can also be used for a more useful purpose. It can be used by police department to find users downloading pornographic material on the Internet. Once the sites serving such material is identified, this scheme can be used to identify movies and catch people even if they use encryption. The police department probably have access to information collected by routers and force routers to collect such information.

4.3 Time Series Representation and Similarity Search

We use time series representation for movie traces and use similarity search to find similar movies. We use the received packet number as a time index and packet size as value for that particular time. In the experiments we used the first 500 packets. Similarity search is a database technique for finding similar items in the
database closest to the given item. Time series data is usually viewed as a point in a high dimensional space and similarity search between the two time series data is computed by using a distance in high dimensional space. The euclidean distance defined by $\|x - y\| = \left(\sum_{i=1}^{n}(x_i - y_i)^2\right)^{1/2}$ where $x$ and $y$ are n-dimensional points (time series data with n attributes) is the most common distance metric used. The traces we used can be considered as 500 dimensional points and the similarity search finds the closest traces in this 500 dimensional space.

There are many issues regarding the scalability and feasibility of such attacks since a trace of a long movie sequence can consist of thousands of packets and finding the closest trace using the trivial methods of one-by-one comparison is too costly. These issues are addressed by the database community. Data size is reduced using methods called *dimensionality reduction*. Typical dimensionality reduction works by transforming data using transformations such as Discrete Wavelet Transform (DWT) or Discrete Fourier Transform (DFT) and retaining small number of leading coefficients. Efficient methods for similarity search have also been developed. Using triangle inequality in higher dimensional space some of the comparisons can be eliminated. Related work on dimensionality reduction and similarity search is discussed in related work chapter.

### 4.4 Experimental Results

We used the web site of BBC\textsuperscript{2} for downloading video clips. This site has hundreds of video clips in Realvideo format. Some of the clips are around 1 minute in length, some are much longer in the range of hours. We streamed a 2-minute prefix of the stock video.

\textsuperscript{2}http://www.bbc.co.uk

31
movies for our experiments. Our client was a Linux PC running Red Hat 8.0 connected to LAN. The client was 15 hops (using traceroute) away from the web server. The route from the client to the server is given in figure 4.4. Note that 15 hops is quite high for streaming sessions. We picked 50 video clips and streamed each video clip 4 times from server to client at different times of day. Streaming all 50 clips takes about 2 hours. We collected the first traces between 9:30 pm and 11:30 pm on February 12. Second traces were collected between 12:30 am and 2:30 am on February 13. The third and fourth traces were collected between 11:00 am and 1 pm and between 3 pm and 5 pm on February 13 respectively. All times are Eastern Standard Time. We used tethereal which is a text-based version of ethtool\(^3\) to collect packet traces.

\(^3\)http://www.ethereal.com
Figure 4.5: Average rank of traces belonging to same video

We had 50 clips and 4 traces for each clip. We put all these traces in a pool and used first 500 packets of each trace in the experiments. We picked a movie trace and found the $l_2$ distance of this trace to all other traces. We sorted the results based on distance. What we were interested in is the rank of traces belonging to the same movie in this sorted distances. It is natural to think that traces belonging to the same movie will be closer to each other in rank. The average, minimum and maximum rank of traces are given in figures 4.5, 4.6 and 4.7 respectively. Out of 200 traces 106 of them have the closest trace belonging to same video (minimum rank is 1). This means with more than 50 % probability closest trace identifies the movie.

In some cases movie traces belonging to same video clip have high distance between them. There are 2 possible causes of this. When a chunk of a movie is lost, similarity search compares different parts of the movie and a high distance is obtained. This
can be minimized by representing movies as time series data and defining $i^{th}$ data point as the sum of sizes of first $i$ packets. We used full search of traces, window based traces can be used as well. A window of packets can be picked and movies that have segments similar to the data in this window can be searched in trace collection. This method is known as the q-gram method in database community. Another reason for high rank between traces of the same movie clip is that, out of 4 clips 2 may be of one quality level and remaining 2 may be of another quality level. Since different quality levels are compared, the distance between them and the maximum rank will be high. There are many possible ways of improving the accuracy of the attack. We did not try to improve the accuracy since our focus is to raise awareness about the information that is revealed by traffic characteristics of media data regardless of whether it is encrypted or not.
A streaming trace depends on 2 things: movie characteristics and available network bandwidth. UDP based streaming algorithms are vulnerable against passive attacks because data sent depends more on video characteristics and less on network characteristics. TCP-friendly streaming algorithms should be preferred to limit passive attacks.

Passive attacks on streaming video will most likely be based on data collected by a router. A router closer to the server on the path can collect more accurate data since some packets might be lost on the way from the router to the client, and similarity search may fail (since not designed for missing data). The closer the router is to the server, the higher the accuracy of passive attacks. Our client was 15 hops away from the server.
The duration of movie clips can be an important factor in identifying the movie. It provides additional information that can be used to find similar traces. Some video clips we streamed were 1 minute in length. Some of them were hours in length and we streamed roughly a 2 minute prefix. We did not use length as a factor in our experiments.

4.5 Prevention of Passive Attacks

In this section we propose a framework based on MPEG coding such that all the movies generate the same trace even when they are transmitted over a network with exactly same conditions. So using this framework even if data is collected by routers, the data is practically useless and can not be used to identify movies.

Secure streaming guarantees that no information is revealed by the encrypted video other than the sender and the receiver. We next discuss secure streaming for variable bitrate and constant bitrate coded video.

There are two ways to handle bitrate for video coding: Variable Bit Rate (VBR) and Constant Bit Rate (CBR). VBR allows compressed bitrate to vary. Scenes with pans, zooms, and fast motion are coded with high bitrate and scenes with little or no motion are coded with low bitrate. In CBR bitrate is constant over some averaging window. The compression ratio achieved by VBR is much better than CBR and VBR offers better quality video using lower bandwidth for streaming applications. However, VBR is more vulnerable since it depends more on characteristics of the video. Our approach to secure streaming is by modifying/editing the video such that all movies produce the same trace when transmitted over a network with exactly same conditions. The trace depends on network conditions and movie characteristics.
Since we have no control over network conditions we manipulate videos so that they have similar movie characteristics to an observer who has access to encrypted content.

We assume all videos are in MPEG format and our secure streaming protocol is based on the following observation.

**Observation 1** *If the GOP size, frame structure and frame sizes of 2 videos are the same then the streaming algorithm will produce the same trace (same packet sizes) when transmitted over a network with exactly the same conditions.*

Finer level information such as slices are not used by streaming algorithms. Most of the MPEG based streaming algorithms in the literature satisfy this observation.

Now we propose a method for coding and modifying video so that the conditions in the assumption related to GOP size, the frame structure and the frame sizes are met. The GOP size and frame structure are parameters to MPEG encoder when video is coded, so these can be easily controlled. For example we can fix the GOP size to 6 and use the frame structure IBBPBB. The next step is to get the same frame size for videos. We add fillers to smaller frames so that $i^{th}$ frame of all the movies are the same. Based on the above assumption, movies coded using this approach will produce the same trace and are indistinguishable using the trace.

We next investigate this overhead using an MPEG model based on random variables.

In our MPEG model I,P and B frames are uniformly distributed over an interval. This interval is based on traces of *Crocodile Dundee* and *E.T.* which have GOP structure IBBPBB and quantization values 8,10,25 for I,P and B frames, respectively. We assume that I frames are identically and independently distributed (iid) over $[1182,13713]$, P frames are iid over $[28,9527]$ and B frames are iid over $[29, 3120]$. The
ranges are based on the minimum and maximum size of frames of *Crocodile Dundee* and *E.T* based on Table 4.1. We assume that different movies are independently distributed with I,P,B distributions given above. We next compute the overhead of using our secure streaming approach. We found the expected percentage of overhead in terms of number of movies based on our MPEG model. The expected overhead is computed by using the expectation of the maximum of random variables. The expected GOP size of a movie with frame structure IBBPBB can be computed using properties of expectation. Since frame sizes are independent of each other in our model, expected GOP size is $E(\text{IBBPBB}) = E(I) + E(B) + E(B) + E(P) + E(B) + E(B)$ which can be written as $E(\text{IBBPBB}) = E(I) + E(P) + 4E(B)$ where $E(I)$, $E(P)$, $E(B)$ denotes expected size of I,P and B frames respectively. The expected GOP size after fillers for $k$ movies can be computed using $E(\text{IBBPBB}) = E(MI) + E(MB) + E(MB) + E(MP) + E(MB) + E(MB)$ which can be written as $E(\text{IBBPBB}) = E(MI) + E(MP) + 4E(MB)$ where MI, MP, MB are random variables which are maximum of $k$ random variables uniformly distributed in the range given above. Expected value of maximum of $k$ random variables uniformly distributed over $[a,b]$ is given as $(a+bk)/(k+1)$. The overhead is defined as extra fillers that needs to be inserted into movies. Overhead is the difference between expected

<table>
<thead>
<tr>
<th>Frame</th>
<th>Crocodile Dundee</th>
<th>E.T.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>I frame</td>
<td>1316</td>
<td>13713</td>
</tr>
<tr>
<td>P frame</td>
<td>29</td>
<td>9527</td>
</tr>
<tr>
<td>B frame</td>
<td>31</td>
<td>3120</td>
</tr>
</tbody>
</table>

Table 4.1: Minimum and maximum frame sizes for Crocodile Dundee and E.T
Figure 4.8: Overhead of secstream

GOP size of a movie with fillers and expected GOP size of a movie without fillers. Overhead is expressed as a percentage of expected GOP size without fillers. Since the same model is used for each GOP, the overhead based on single GOP will be the same as the overhead for long movies as long as movies consist of same number of GOPs. The overhead is given in figure 7.14. Overhead is based on fillers transmitted to hide movie characteristics assuming all movies are equally likely. This overhead can be reduced by editing movies to reduce GOPs that cause large fillers in other movies. Note that fixing the GOP size and frame structure reduces the overhead since I, P and B frames correspond to the same type of frames in all the movies. As shown in table 4.1, B frames have much smaller size than I frames and using fillers to increase size of a B frame to that of an I frame will require huge overhead. Analysis of the movies Crocodile Dundee and E.T shows that 90 % of the I frames have size
in range [1800-8000], 90% of P frames have size in range [25-2250], and 90% of B frames have size in [25-400]. Frames that have size larger then these can be edited to reduce the size of the frame. The above scheme guarantees that no information is revealed even if movies are transmitted over a network with exactly same conditions. Typically network conditions will vary on congestion, time of day etc. Overhead can be reduced by padding only very small frames up to some limit depending on frame type. Peaks and dips in time series representation are the parts that reveal the most information so by reducing size of peaks and increasing dips we can reduce the possibility of the movie being identified. But in this case the guarantee of no information being revealed is lost, everything is dependent on network conditions.
CHAPTER 5

LOSSLESS VIDEO TRANSMISSION AND SECURITY MECHANISM

As networking and computing technologies continue to advance, the ability to support advanced video-based applications that deliver high-quality video is now possible. With the large amount of research focused on the efficient coding, storage, retrieval, and transmission of digital video, we are currently able to support DCT-based video streaming across networks fairly well. We are, however, still unable to support applications that require more stringent bounds on the quality of the video such as medical imaging, advanced scientific visualization, and computer graphics.

Unlike *lossy* DCT-based video compression algorithms that transform the data into the frequency domain so that it is more compressible, lossless image compression techniques attempt to predict the value of each pixel based on some of the surrounding pixels and then entropy encoding the difference to the predicted value. More importantly, it has been shown that there is little advantage to temporal compression of video sequences [49]. Because of this, the compression ratios achieved by lossless image compression standards are much less than that of DCT-based algorithms (typically 2:1 for lossless and 25:1 for DCT-based). Thus, the coding and transmission of lossless video can be extremely resource hungry.
Supporting lossless video-based applications is extremely difficult because they require low-latency delivery of video while requiring much larger bandwidth, generally considered two mutually exclusive goals. In our discussion with these researchers and scientists, two main themes manifested themselves. First, they need the ability to have low-latency visualizations so that they can understand what is happening in the simulation data at a higher-layer (that is, general understanding). Second, they need the ability to query the data to be able to extract meaningful information from the data accurately. Note that the image data is not necessarily confined to 24-bits per pixel.

We propose a lossless video compression technique that transmits lossless components on-demand and has comparable and better compression ratio to current lossless image compression techniques as long as percentage of lossless requested frames is less than 50\% but, more importantly, makes the data more amenable to network transmission. The main purpose of this approach is to provide (i) a low-bitrate MPEG video sequence that supports the low-latency requirements, allowing the researchers to understand the data at a higher-level and (ii) a lossless or loss-bounded differential file that allows the original image data to be reconstructed, if necessary. Our results show that we can achieve lossless and loss-bounded image compression that has comparable performance to lossless video compression algorithms but also supports low-latency delivery (through the smaller MPEG file). The experiments use two video sequences, one from medical imaging and the other from a scientific visualization program.
5.1 Proposed Method

To aid scientists in the visualization and understanding of scientific data, we propose a multi-layer lossless video compression algorithm that supports low-latency viewing of time-varying data and supports lossless query retrieval of the original image data. The basis of our approach is to use MPEG to compress the video stream to create a basic video that provides the scientist with a high-level, real-time view of the visualization. After the scientist understands the high-level view of the visualization and wants to look at actual values of pixels (which may represent physical phenomena such as pressure) we augment the data with a lossless differential that provides the original data back to the user (i.e. is lossless). For clarity, we refer to the lossless differential as an enhancement and use differential to mean that a frame is differentially coded with respect to a previous frame.
In the general scenario where millions of people may want access to the video, this type of multi-layering may not be possible. The following are important issues for our system.

- Lossless differences are based on a specific quality stream. Quality adaptation in the network should not be used unless it is supported by multiple differences based on quality.

- MPEG implementations that use shortcuts to optimize for speed should be avoided since they result in different image data. System should be implemented using MPEG-compliant decoders. Following are examples of shortcuts: Different inverse-DCT algorithms being implemented in the decoder for performance [71, 17]. The decoder may not perform all the calculations [34].

The main reason for using an MPEG based layer as a building block is to make our system flexible in many ways. Some of the advantages include:

- Existing streaming algorithms and protocols proposed for MPEG can be used, MPEG players require minimal modification to support our scheme.

- Security in the form of encryption and authentication can be achieved by encrypting, authenticating only the MPEG stream [67]. Differences do not have any information value without MPEG file and need not be encrypted. This reduces security cost significantly.

The encoding phase of our proposed approach consists of four main steps: (a) MPEG video compression, (b) MPEG video reconstruction, (c) differential image calculation, and (d) differential entropy encoding. In the first step, we encode the
Figure 5.2: Snapshot of system based on MPEG-1

video frames using quantization levels of 1 for the I, P, and B frames. Using a low quantization level, ensures that the artifacts introduced during the DCT transformation remain small. This is particularly important when dealing with computer generated data that contain many high frequency components. In the second step, we decode the video frame to determine the resultant video frame. This decoded frame is then used to create the enhancement image from the original frame in the third step. In the final step, we lossless compress the enhancement video frame. As we will show in the experimental section, Huffman compression or LZ and not LOCO-I will be the most effective in compressing the enhancement because it provides similar levels of compression and provides much faster decoding speed. A block-level diagram of the compression stages is shown in Figure 5.1.
Viewing the compressed data occurs in several phases as well. Initially, when the scientist is trying to understand the simulation, the highly compressed MPEG-file is streamed to the user. If the user finds an interesting phenomena that he/she wants to look at in greater detail, the lossless compressed enhancement is transmitted to the user. The enhancement is then decoded and added back to the MPEG video frame to create the original video data without loss. In many cases, it is expected that the user will not need to see the lossless compressed video frame, resulting in a large savings of network bandwidth. A snapshot of the system based on MPEG-1 is shown in figure 5.2. When the user sees something interesting he can pause the video and then switch to lossless mode. This will send a request to server asking for compressed differences for that frame. Upon receiving differences, player adds then to the picture. When the user switches to lossless mode, we expect the user to spend a couple of seconds examining it and we believe latency will not be a major issue. In lossless mode, user can view previous frame and next frame by using rewind and step buttons.

For high resolution images such as the 1920 by 1035 original wind images that we had, sending all the differences is not feasible since the file size for each image is in megabytes and user will be interested in a specific part of the image anyway. We support region of interest by tiling the large images into 4x4 grid of tiles. The user picks a rectangular area on the image, and notifies the server. The server needs to send only the tiles that intersect user’s query.
5.2 Experimentation

In this section, we compare and contrast the performance of our scheme with other algorithms such as LOCO-I for lossless video compression and transmission. We begin with a description of the experimental setup including the data sets used, we then investigate our scheme in detail.

For our experiments, we used two different data sets. The first data set is a scientific visualization simulation that shows the movement of clouds and winds around the earth. The original data set consists of frames that are 1920x1035 pixels in size. We extracted a 352x240 pixel subregion within the data. In addition, we used all 45 frames from the data set in our experiments. We refer to this data set as the wind data set. The second data set is a visualization of a rotating head. We had 72 images that show the 360 degree camera rotation around the head. We refer to this data
set as the *head* data set. Sample images from *head* and *wind* data sets are shown in Figures 5.3 and 5.4 respectively.

We used two publicly available MPEG encoders and decoders for our experiments. The first set consists of the Berkeley MPEG tools *mpeg_encode* (version 1.5) and *mpeg_play* (version 2.3).\(^4\) The second set of software consists of the MPEG Software Simulation Group tools *mpeg2encode* and *mpeg2decode*.\(^5\) Experimental results presented in this section are based on MPEG Software Simulation Group tools.

For our experiments, we coded several basic compression algorithms and were able to gather some publicly available image coding algorithms. The code for the comparison includes:

**LOCO-I:** Images are compressed independently using LOCO-I, a publicly available

LOCO-I encoder and decoder from the Univ. of British Columbia

\(^4\)These are available via the web at www.bmrc.berkeley.edu

\(^5\)These are available via the web at www.mpeg.org
MPEG + **LOCO-I enhancement**: implements the MPEG video plus enhancement frames with respect to the MPEG stream using LOCO-I compression.

MPEG + **Huffman enhancement**: implements the MPEG video plus enhancements that are lossless compressed using Huffman compression.

MPEG + **LZ enhancement**: implements the MPEG video plus enhancements that are lossless compressed using *compress* which is an adaptive Lempel-Ziv compression.

MPEG + **PNG enhancement**: implements the MPEG video plus enhancements that are lossless compressed using PNG (Portable Network Graphics).

**LOCO-I Differential**: uses modified LOCO-I encoder to do differential coding of video frames (from a previous frame). The purpose of this algorithm is to show the effect of temporal encoding.

For our research, we are interested in following questions:

- What overall compression ratio is achievable with the various algorithms?
- How do they compare in terms of compression and decompression time?
- How good is the quality of MPEG file generated?
- When is our approach preferable to one layer lossless video coding?

In the rest of this section, we will describe the experiments that we ran and the resulting data that was obtained.
Figure 5.5: Compression ratio of frames for head data for lossless compression

Figure 5.6: Compression ratio of frames for wind data for lossless compression
Figure 5.7: Compression ratio of frames for head data for loss-bounded compression with bound 2

Figure 5.8: Compression ratio of frames for wind data for loss-bounded compression with bound 2
Figure 5.9: Compression ratio of frames for head data for loss-bounded compression with bound 4

Figure 5.10: Compression ratio of frames for wind data for loss-bounded compression with bound 4
5.2.1 Compression Performance

To test the compression performance of the algorithms, we ran the sample data sets through the various algorithms and plotted the resulting compression ratios (on a frame-by-frame basis). The results for lossless compression, loss-bounded compression with bound 2 and loss-bounded compression with bound 4 for head data are given in Figure 5.5, Figure 5.7, and Figure 5.9 respectively and the results for wind data are given in Figure 5.6, Figure 5.8, and Figure 5.10 respectively. Compression ratio of MPEG file is not included in these graphs we will discuss overall compression ratio later. As shown by the figures, the compression ratio achieved by the various algorithms is very similar. The wind figures show that there is some advantage to doing differential compression using LOCO-I. However, this advantage becomes a disadvantage when applied to the head data set. The reason for this is that the wind data set has a background that does not change and the actual wind vectors slowly change. As a result, the data between frames is very similar. The main disadvantage is that for sequences that have larger motion (as in head), performing differential lossless compression is terrible. The main reason is that the differential is “fixing” data from the previous frame, resulting in a higher entropy. This experiment verifies the previous finds why the application of differential coding for lossless video is difficult [49].

In these figures, we also see that using MPEG and Huffman compression on the enhancements (between the MPEG and actual frame data) results in compression performance close to that using LOCO-I only or using MPEG and LOCO-I compressed enhancements. Finally, we note that with looser error bounds on the compression that the ability to compress the data becomes easier.
In Table 5.1, we have graphed the overall compression ratio for the various algorithms and loss-bounds. Compression ratio is computed by dividing the size of the original images by the size of compressed images. For algorithms which also have an MPEG part the size of compressed images is sum of size of MPEG and size of compressed differences. We implement lossbounded compression using Huffman by mapping many nodes to one node without violating lossbound and then we compress the differences using this smaller tree. LOCO-I(orig.) is compression of original images using LOCO-I instead of the differences. As shown in the figure differences compress better than the original, especially for loss bounded compression. For high quality MPEG streams differences compress better than original since displayed image is close to original.

5.2.2 Algorithm Speed

So far, we have shown that the proposed approach results in similar compression to applying LOCO-I to each frame individually. We next compare the performance of our approach to LOCO-I. In Table 5.2, we show the time required to compress and decompress the two data sets. Here, we see that MPEG compression is the most expensive operation, due mostly to the expensive motion compensation step. We also see, however, that the Compress program on Unix has the best speed (probably due to heavy optimization), one-fourth to one-sixth of LOCO-I but typically has lower compression ratio. MPEG and Huffman compression result in a total decompression time that is one-half to one-third that of LOCO-I. As a result, we will be able to deliver the same quality video with less overhead at the client. The reason LOCO-I is so slow is that performs a number of operations per pixel in both the encoding and decoding
Table 5.1: Overall compression ratio

<table>
<thead>
<tr>
<th>Data</th>
<th>Comp.</th>
<th>MPEG+ LOCO-I (orig.)</th>
<th>MPEG+ LOCO-I</th>
<th>MPEG+ HUFFMAN</th>
<th>MPEG+ LZ</th>
<th>MPEG+ PNG</th>
<th>LOCO-I</th>
<th>DIFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>lossless</td>
<td>1.70</td>
<td>1.77</td>
<td>1.53</td>
<td>1.54</td>
<td>1.67</td>
<td>2.32</td>
<td>2.95</td>
</tr>
<tr>
<td></td>
<td>bound-2</td>
<td>2.67</td>
<td>2.92</td>
<td>2.63</td>
<td>N/A</td>
<td>N/A</td>
<td>4.33</td>
<td>3.44</td>
</tr>
<tr>
<td></td>
<td>bound-4</td>
<td>3.10</td>
<td>3.52</td>
<td>3.22</td>
<td>N/A</td>
<td>N/A</td>
<td>5.59</td>
<td>4.42</td>
</tr>
<tr>
<td>Wind</td>
<td>lossless</td>
<td>1.57</td>
<td>1.62</td>
<td>1.40</td>
<td>1.26</td>
<td>1.53</td>
<td>1.94</td>
<td>2.94</td>
</tr>
<tr>
<td></td>
<td>bound-2</td>
<td>2.68</td>
<td>2.82</td>
<td>2.33</td>
<td>N/A</td>
<td>N/A</td>
<td>3.71</td>
<td>6.41</td>
</tr>
<tr>
<td></td>
<td>bound-4</td>
<td>3.22</td>
<td>3.47</td>
<td>3.07</td>
<td>N/A</td>
<td>N/A</td>
<td>4.88</td>
<td>9.18</td>
</tr>
</tbody>
</table>

Table 5.2: Encoding/Decoding performance

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Encoding Time (Sec)</th>
<th>Decoding Time (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Head</td>
<td>Wind</td>
</tr>
<tr>
<td>intra-frame LOCO-I</td>
<td>8.12</td>
<td>10.20</td>
</tr>
<tr>
<td>LOCO-I differences</td>
<td>8.52</td>
<td>10.51</td>
</tr>
<tr>
<td>MPEG</td>
<td>41.91</td>
<td>36.29</td>
</tr>
<tr>
<td>Huffman differences</td>
<td>5.40</td>
<td>6.18</td>
</tr>
<tr>
<td>LZ differences</td>
<td>2.22</td>
<td>2.75</td>
</tr>
<tr>
<td>PNG differences</td>
<td>21.57</td>
<td>23.25</td>
</tr>
</tbody>
</table>
phases. These steps in encoding include: predicting the value based on neighboring pixels, calculating a prediction based on gradient information between neighboring pixels, and adaptively coding the value using Golomb codes. The decoding must perform these steps in the reverse order.

5.2.3 Effect of MPEG file size on Overall Compression Ratio

For each data set we created 3 different MPEG files with different inter and intra quantization step sizes. We investigated the PSNR for the MPEG files as well as the frame-by-frame compression ratio of the differences. PSNR is computed using the formula

$$PSNR = 20 \log_{10}(255/RMSE)$$
Figure 5.12: Compression ratio of wind dataset

Figure 5.13: SNR of head dataset
Figure 5.14: Compression ratio of head dataset

Figure 5.15: Single layer lossless video compression vs our scheme on head data
Figure 5.16: Single layer lossless video compression vs our scheme on wind data

where RMSE denotes root mean squared error. The results for Wind data are given in figures 5.11 and 5.12, results for Head data set are given in figures 5.13 and 5.14 respectively. MPEG sequences with high PSNR have higher compression ratio for enhancement frames since base layer images are closer to original images. Lower quality MPEG sequences result in higher overall compression ratio, but in this case users are more likely to request frames lossless and amount of data transmitted end up being higher. When all frames are requested lossless our scheme will have lower overall compression ratio than individual frame-by-frame LOCO-I. For a given percentage of lossless frames we investigated the compression ratio a single layer lossless video compression algorithm should have in order to transmit same amount of data over the network. Graphs for the Wind and Head data sets are given in figure 5.15 and 5.16 respectively. As can be seen from the figure, If low percentage of frames are requested
Figure 5.17: Entropy of differences for head images with lossless compression

lossless then our scheme is better since lossless video compression algorithms can not reach such compression ratios.
Figure 5.18: Entropy of differences for wind images with lossless compression

Figure 5.19: Entropy of differences for head images with lossbound 2 compression
Figure 5.20: Entropy of differences for wind images with lossbound 2 compression

Figure 5.21: Entropy of differences for head images with lossbound 4 compression
Figure 5.22: Entropy of differences for wind images with lossbound 4 compression

5.2.4 Entropy of Differences

We also investigated the entropy of differences for the decoded MPEG file from the original images. The entropy based compression ratios for the two sample sequences are shown in figures 5.17 through 5.22 for lossless, loss-bound 2, and loss-bound 4 compression, respectively. As shown in these figures, using lossless compression Huffman results in entropy-based compression ratio and as the differences increase as the loss bound increases. As in theory, the compression ratio of Huffman is below the compression ratio computed using entropy. When the loss bound is higher, the Huffman compression ratio falls much below the entropy-based limit. This is because the probabilities of symbols are disproportionate and the highest probability symbols have 1 bit length when compressed. The Huffman compression ratio reaches the entropy-based limit only when the probabilities of symbols are 1/2, 1/4, 1/8, etc.
5.3 Security Mechanism for Layered Lossless Coding

In proposed layered lossless video coding framework, the size of MPEG layer is about 10-15 % of the total size. The lossless differences does not have a meaning without the MPEG base layer. So we encrypt only the MPEG layer and send lossless differences unencrypted. It is easy to alter a Huffman coded file and get a valid Huffman coded file. In many applications it is very important to get lossless video and the application should guarantee that the image is original before any action is taken. Lossless differences are authenticated to make sure they are not altered by a third party. Authentication is a much more efficient operation than encryption so authentication does not incur much overhead. Authentication is done using the cryptographic hash function MD4. MD4 takes the frame as input and generates a 20 byte message digest. This message digest is signed with private key of server and sent to client. Clients compute hash function on the received frame and compare the result with the signed value, if they match then the frame is authentic.

We investigated whether any information can be extracted from the lossless differences. Most of the differences will be small single digit values. Typical information that can be left in differences is edge information. Edges are troublesome for compression and many lossy compression algorithms will do a poor job in edges. A sample difference image belonging to wind dataset is given in figure 5.23.
Figure 5.23: Lossless differences for sample wind image

Figure 5.24: Canny edge detection on lossless differences
Figure 5.25: Canny edge detection on original image

Edge detection is a problem of fundamental importance in image analysis. In typical images, edges characterize object boundaries and are therefore useful for segmentation, registration, and identification of objects in a scene. Many methods are developed for edge detection including methods based on gradients, LOG edge detection and canny edge detection. Edge detection methods are relatively easy to compute and can be automated. If lossless images are sent unencrypted, it is not feasible to use a human to look at lossless images or restored lossless images to see what is in the movie. A feasible attack should be automated. An automated attack can filter data and keep what seems interesting.

We used canny edge detection to find the edges in the above lossless difference image. Canny edge detection is regarded as one of the best edge detection algorithms. The edge image is given in figure 5.24. We also used canny edge detection on original image to see how they compare. Edge image on original frame is given in figure 5.25.
Original image has a map of Australia covered with clouds. The edges of map of Australia is visible in edge detected original image. As can be seen from the figures edge detection does not work on lossless differences. Most of the values in lossless differences are low single digit numbers. When 2 neighboring pixels differ in by 4 in original image, this not enough to consider it as part of an edge. But in lossless differences, 2 pixels differing by 4 are considered as part of edge since most of the numbers are low. So too many edges are detected by the edge detection algorithm. This is a good sign of lack of meaningful information in lossless differences.
CHAPTER 6

SECURE VIDEO TRANSMISSION USING PROXIES

Increase in commercial use of the Internet has resulted in great demand for multimedia. Multimedia data however poses some challenges due to its unique characteristics. Multimedia data sizes are typically very large (in GBs) and multimedia data needs to be processed in real time. Managing huge amounts of data requires faster processors (for decoding), more storage space and more bandwidth (for transmission). There are a couple of factors which will determine future of commercial multimedia applications. Efficient streaming mechanisms, Digital Rights Management (DRM) products are some of the factors.

Over the last couple of years there has been a lot of work in the area of bandwidth smoothing to reduce the burstiness of compressed video in video-on-demand architectures. Using a buffer at clients to store prefetched video, and collaboration of client and server to compute transmission plans helps achieve bandwidth smoothing. Characteristics of networks with which the clients are connected can vary greatly in bandwidth and processing power. The available data rate can range from a few kbps in a GSM link up to 100Mbps in a fast ethernet network. Users with similar connection can experience much lower data rates because of congestion. One possible solution to this heterogeneity problem is to use proxies between the client and server.
Proxies can shape the video based on client resources. For MPEG-1 video shaping can be in the form of frame dropping to reduce frame rate or transcoding to reduce quality. Proxies are also used for caching purposes by the multimedia community.

DRM has become an important issue for content distribution over the Internet. However proposed solutions to DRM do not provide schemes specific to video. Such solutions will not be scalable considering the large data size and delay requirements of video data. To provide DRM for video these characteristics should be taken into account. We also note that the same scheme can not be used for audio files in mp3 format (typically 3-4 MB) and video files in MPEG format (typically in GB), and that DRM schemes so far have focused on data files and audio files in mp3 format. Furthermore, encryption algorithms such as DES and RSA are not suitable for video data due to the size of data that needs to be encrypted. Light-weight encryption mechanisms have been proposed for video data and these algorithms can encrypt data much faster than traditional algorithms. For video-on-demand servers that need to encrypt many video streams in real-time light-weight encryption is preferable.
In this chapter I investigate the feasibility of end-to-end security in proxy-based systems. In proxy-based systems there is a proxy between server and client and this proxy aids in streaming by adapting quality of video to available resources. Network model for proxy-based systems is given in figure 6.1. We focus on encryption and authentication and investigate whether end-to-end solutions exist. Without end-to-end solutions the sequence of steps for encryption is as follows: proxy has to decrypt the data, perform proxy operation and then reencrypt the data and send to client. The ultimate goal in end-to-end security is to perform all proxy operations (frame dropping, quality adaptation, caching) in the encrypted domain. Similarly we want to be able to perform proxy operations without violating the authentication so that client can easily authenticate the video. Authentication is typically easier because cryptographic hash functions can be used for authentications. We considered 3 types of proxy operations: frame dropping, quality adaptation in the form of transcoding and caching. Our results show that for frame dropping and caching can be supported in encrypted domain. For authentication all three of these operations can be efficiently supported.

6.1 End-to-end Encryption

We next investigate feasibility of end-to-end encryption for proxy-based video dissemination.

6.1.1 Frame Dropping

MPEG video consists of I, P and B frames. If there is not enough bandwidth B frames can be dropped and the I and P frames can still be displayed. In this section,
we focus on dropping frames in encrypted MPEG streams. The main result is that it is possible to drop frames in encrypted form using block ciphers.

Blocks ciphers [61] divide the data into blocks of fixed size and then encrypt the blocks independently. This allows decryption even if some blocks are missing. Block ciphers are preferable because of this property.

In this scenario, assume that we have a block cipher with blocks of 128 bytes. Further, assume we have stored video and we know the frame sizes. If the proxy knows the frame sizes it can adaptively alter the transmission rate to meet network conditions by dropping entire streams.

Consider the example given in figure 6.2. The first row indicates the original file, the second row indicates encrypted file, the third row indicates frame dropped encrypted file and the fourth row indicates the decrypted file. Finally the fifth row is MPEG file after postprocessing. In the figure, \( x \) denotes the start of a frame and rectangles denote the encryption block. Let us assume that we want to drop frame 2. Frame 2 is divided into 3 blocks. The third block has the end of frame 1 and the start

---

**Figure 6.2: Frame dropping in encrypted stream**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

original stream
encrypted stream
encrypted stream
decrypted stream
postprocessed stream
of frame 2 and the fifth block has end of frame 2 and start of frame 3. If encrypted using standard techniques, we can not identify information belonging to frame 1 in block 3 without decrypting it.

We next provide two methods for dropping frames in encrypted domain and evaluate their performance.

**Strategy 1**

This strategy does not do any decryption at proxy, it drops the blocks that can be dropped without problem. In the figure only block 4 can be freely dropped because block 3 has some information belonging to frame 1 and block 5 has some information belonging to frame 3. Note that in this case we are not able to drop as much information as we like. As long as the frame sizes are much higher than block sizes amount of information we are not able to drop is very low compared with dropped information. In this strategy when receiver decrypts the blocks it should drop information belonging to frame 1 in block 3 and information belonging to frame 2 in block 5. So a postprocessing step is needed.

**Strategy 2**

In this strategy, the server sends the information it makes sure that each frame starts in a new block. Obviously, compression algorithm may need to be modified for this. Another possibility is for the server to add some filler that so that next frame starts in a new block (player needs to ignore the filler). This strategy makes the proxy operation much more easier. The proxy just drops the blocks containing encrypted streams of dropped frames.
6.1.2 Quality Adaptation

One would like to combine the virtues of security and the network adaptability in proxies. In this section, we show that, in general, this is not possible with coefficient-based encryption and that it is necessary to decrypt and re-encrypt video for quality adaptation.

The discrete Cosine Transform (DCT) is used in MPEG. After the DCT transformation, the coefficients are quantized (divided by quantization value to get smaller coefficients and hence better compression). DCT coefficients are taken in zig-zag order, compressed using run-length encoding and then compressed using Huffman compression. Transcoding is typically used for quality adaptation, which means that the quantization value is increased (coefficients become smaller and compress better). To do transcoding, Huffman decompression and run-length decompression are done to retrieve the coefficients. Then the new coefficients are computed and are compressed.

If the MPEG file is encrypted using an encryption algorithm such as DES the only way to do transcoding is to decrypt the data, do the transcoding and, re-encrypt the data. This requires that the server knows the existence and identity of the proxy and that the proxy knows the identity of the client.

In this section, we investigate whether transcoding can be done on encrypted coefficients (without decryption). When we encrypt coefficients, we have to make sure that they can still be compressed well. We will focus on two encryption schemes: one that encrypts each coefficient independently and one that encrypts the whole block of coefficients.
Encryption of Coefficients Independently

Assume that we encrypt each coefficient separately. Let $e(x)$ be the encryption of coefficient with value $x$ and $q(x)$ be the quantized value of coefficient with value $x$. We want an encryption algorithm which commutes with quantization. We would like to get the same result when we do quantization in encrypted form and encryption after quantization. Thus we need $e(q(x)) = q(e(x))$. Quantization can be represented by multiplication by a constant so $q(x) = c \cdot x$ where $c$ is a constant that determines quantization. Substituting this into the equation we get $e(cx) = c \cdot e(x)$. Assuming $c$ and $x$ are from the same domain, the functions that satisfies the condition are of the form $e(x) = k \cdot x$ where $k$ is a constant. More formally

**Theorem 1** Let $f : \mathbb{R} \rightarrow \mathbb{R}$, $f(cx) = c \cdot f(x) \forall c, x \in \mathbb{R}$ then $f(x) = k \cdot x$, $k \in \mathbb{R}$

**Proof 1** Immediate from basic functional analysis.

Note that this multiplication can be done based on the key. For example if corresponding bit of the key is 1 then it is done, if the corresponding bit is 0 it is not done.

Theorem 1 gives us an idea of what kind of functions commute with quantization. Now let us concentrate on a more accurate model. DCT coefficients are integers and $\lceil \rceil$ and $\lfloor \rfloor$ functions need to be used for quantization. Quantization is based on the sign of the coefficients. If coefficients are positive, they are rounded down after quantization. For example 2.3 is rounded to 2. If coefficients are negative, they are rounded up after quantization. -2.3 is rounded to -2. Thus, a accurate equation is given below.
Figure 6.3: Original frame

\[
e([x/c]) = \begin{cases} 
\left[ \frac{e(x)}{c} \right] & \text{if } e(x) > 0, x > 0 \\
\left[ \frac{e(x)}{c} \right] & \text{if } e(x) < 0, x > 0
\end{cases}
\]

\[
e([x/c]) = \begin{cases} 
\left[ \frac{e(x)}{c} \right] & \text{if } e(x) > 0, x < 0 \\
\left[ \frac{e(x)}{c} \right] & \text{if } e(x) < 0, x < 0
\end{cases}
\]

The only solutions to this equation are \( e(x) = x \) and \( e(x) = -x \). Note that the equations should hold for every positive integer \( c \).

The algorithm proposed in [16] flips the sign bits of coefficients based on a key. It conforms to the rule derived above, encryption of the algorithm satisfies \( e(x) = -x \), Therefore it can be used for transcoding in the encrypted domain.

We believe that flipping the bits of DCT coefficients is not a very secure algorithm. Most of the information will be in the DC value and the first few AC coefficients. One possible attack is try to find the sign of these coefficients. Most of the coefficients (especially the DC and the first few AC coefficients) will be positive, an immediate
Figure 6.4: All DCT coefficients flipped to positive

attack is to make signs of all coefficients positive and view the image. First frame of flower garden sequence and this attack is shown in Figures 6.3 and 6.4.

Encryption of Whole Block

Another possibility for encryption is to use a linear transformation which will map DCT coefficients onto another set of coefficients. The Zig-zag permutation algorithm which has been proven to be weak for encryption uses a linear transformation. There are some requirements of linear transformations that can be potentially used for encryption so that encrypted block compressed well. First, AC coefficients will typically become smaller and smaller when considered in zig-zag order, the linear transformation should not change the order. DCT coefficients will have many trailing zeros after quantization and Huffman coding of coefficients are optimized for this. The second requirement is that zeros should be mapped to zeros. These requirements
are only a subset which prevents the performance of compression from being worse when coefficients are encrypted. More formally,

**Theorem 2** Let $f : \mathbb{Z}^n \to \mathbb{Z}^n$ and let $x \in \mathbb{Z}^n$ and $y = f(x)$ st

1. $x_i \leq x_j \Rightarrow y_i \leq y_j \forall x, y$

2. $x_i = 0 \Rightarrow y_i = 0$

then $f(z) = k * z, \ k \in \mathbb{Z}, x \in \mathbb{Z}^n$

**Proof 2** Immediate using linear transformation theory (linear operator theory).

This class of transformations is the same as the set which encrypts each coefficient separately ($f(x) = c * x$). Thus, the only possible values for $c$ are integer values and as coefficients are multiplied by a larger integer, the file size becomes larger. There are only a few possible values of $c$ that will make sense and those will not provide security. By enumerating possible values of $c$, we can find out what the image looks like.

Based on the above discussion we believe trying to find encryption algorithms that can survive transcoding are not a good idea. Instead a light-weight encryption algorithm needs to be used for encryption and decryption.

### 6.1.3 Secure Proxy Caching

Typically, caching is difficult to support with encryption. The main issue is whether to cache video data in decrypted form or in encrypted form. Caching in decrypted form seems more intuitive but in this case we lose the advantages of proxy operations in encrypted form such as frame dropping. Let us analyze the caching options in more detail.
Caching in Decrypted form

In this case, if we have a cache hit we need to encrypt data and send it to client, if we have a cache miss we send a request to the server and when the data arrives, send it to user and decrypt and place the data in cache. Overall, we need to do either encryption or decryption for each data frame. If repeated misses occur then the decrypted data is replaced repeatedly and resources used for decryption are wasted. Security cost at the proxy can be represented as $h * c_e + m * c_d$ where $h$ is hit rate, $m$ is miss rate, and $c_e$ and $c_d$ are encryption and decryption cost respectively.

Caching in Encrypted form

First, let us concentrate on what happens if some encryption algorithm such as DES is used. If the encrypted data is cached, in case of a cache hit, this data has to be retrieved, decrypted and encrypted with the key of new client. In case of a cache miss, data received from server is sent to the client and stored in the cache in encrypted form. So encryption and decryption is done for each hit. Thus the security cost at the proxy can be represented as $h * (c_d + c_e)$. Assuming encryption and decryption cost is the same (e.g. DES), we have $h * c_d + h * c_e$. By comparing the equations for encrypted form and decrypted form we see that caching in encrypted form is better if $h \leq m$.

The above scheme looks like it is difficult to improve, but there is a possible improvement that can be utilized based on the choice of the encryption algorithm. The key question here is can we do decryption with one key and encryption with a second key efficiently. This can be done if encryption algorithms forms an algebraic group.
Theorem 3. Let $G$ be a group, let $a, b \in G$ then

$$a^{-1}b \in G$$

Proof 3. Immediate from definition of group.

Permutation-Complementation (PC) encryption algorithms form a group, therefore we can use the PC algorithm for proxy caching of video. Using PC for caching lets us reduce the cost of secure caching (decryption + encryption) to half by doing a combined operation that we may call transcription. Thus the security cost is $h \cdot c_e$. This is better than caching in decrypted form in all cases. To encrypt the data we have to make at least one pass over the data. Therefore we may argue that this operation is optimal.

6.2 End-to-end Authentication

End-to-end authentication is somewhat easier to support using digital signatures. In a system involving only server and client, server computes the value of cryptographic hash function (typically about 20 bytes) and appends the signed (encrypted using private key) hash value to the message. Upon receiving the message the client computes the hash value on the message, decrypts the signed hash and compares them. If they are equal then the message is authentic. Since it is highly unlikely to find 2 different messages with same hash value, equality of hash values is enough for authentication. When we introduce the proxy in between, this scheme fails. Server inserts signed hash but if the video is transcoded then signed hash will not match the data sent. We need to modify the scheme to handle end-to-end authentication.

Authentication using server-to-proxy and proxy-to-client authentication can be supported as follows. Server adds authentication data (signed has) to video and sends
to client. Client verifies that video is from the server, performs proxy operation, adds authentication data to video and sends to client. In this case, client can verify that video is from the proxy but client can not verify that video originated from the server. In this system server trusts the proxy, client also trusts the proxy since client believes video originated from server. We want to support end-to-end authentication where client can verify that video originated from the server. End-to-end authentication will also reduce processing at proxy since cryptographic hash computation can now be avoided.

6.2.1 Proposed Authentication Scheme

Server first determines a fixed number of quality levels for video. Four to five quality levels will work fine in most video streams. Proxy can only transcode original video to one of the predetermined levels original video being one of the levels. Since we work with stored video server can determine quality levels based on statistical characteristics of the video. Transcoding is done on a GOP basis. Server will add signed hash value corresponding to all the quality levels after each GOP. Hash values are typically 16-20 bytes depending on hash function used. Hash functions such as MD4, MD5 have 16-byte hashes and functions such as SHA has 20 byte hash values. So
for 5 quality levels with SHA has function, after each GOP there will be 5 hash values occupying 100 bytes. Proxy will transcode the video to one of the predetermined quality levels and keep the corresponding signed hash with the transoded GOP as shown in figure 6.5. Upon receiving the video, the client then computes the hash function for each GOP and compares the result with the signed hash stored with the GOP.

In the above scheme, the server computes hash values for multiple quality levels. This will need transcoding at the server for each level. Since hash function depends on the movie only and does not involve any client specific information, we can compute the result of hash function in advance and store it with the movie. So the server reads the movie from the disk, signs the hash values and sends them to the proxy. This does not introduce much overhead for the server. Since server handles multiple clients, schemes with low overhead are desirable. In this scheme server has all the power, server determines the quality levels and proxy can only pick from one of the levels. If proxy picks a quality level not determined by server, computation of cryptographic hash will result in a different value than the signed one and the client will drop that segment of video.

Proposed scheme introduces minimal overhead at the proxy. After transcoding the video, the proxy picks one of the signed hashes and drops the other hashes. Considering the fact that proxies handle multiple clients, this is very good.

From the client perspective proposed end-to-end authentication scheme is no different than direct server-to-client authentication. Client receives the video and verifies authenticity using the signed hash value.
The above discussion shows that the proposed scheme introduces minimal overhead at the server, proxy and client. We can support both frame dropping and transcoding within the same framework. To drop B-frames we can define a quality level without B-frames and use the proposed framework. P-frames can be dropped by defining a quality level without P-frames.

6.3 Experimental Results on Encryption

In this section we provide results to justify the feasibility of our approach. Effects of caching are not investigated in detail since they were analyzed in the previous section. Any algorithm used for caching (with cache replacement policy, actions to be done on hit, miss) can be incorporated into our cache structure. Quality adaptation is typically done at the server in current systems, so our design does not require drastic changes to support it.

We used four MPEG files which have been heavily used for research purposes. The files we used are flower.mpg, bus.mpg, tennis.mpg and twister.mpg.

6.3.1 Frame Dropping

Let us first concentrate on the overhead of frame dropping in the encrypted domain.

Strategy 1

This scheme drops the blocks which consist only of the frames to be dropped. Blocks that partially contain information about frames to be dropped are preserved. Because of this restriction, we may not be able to decrease bandwidth requirements as much as we would like. Some of the important issues are the frame sizes and

82
<table>
<thead>
<tr>
<th>Filename</th>
<th>I frames</th>
<th>P frames</th>
<th>B frames</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number</td>
<td>Avg. Size</td>
<td>Number</td>
</tr>
<tr>
<td>flower.mpg</td>
<td>10</td>
<td>17321</td>
<td>40</td>
</tr>
<tr>
<td>bus.mpg</td>
<td>10</td>
<td>13510</td>
<td>40</td>
</tr>
<tr>
<td>tennis.mpg</td>
<td>26</td>
<td>24089</td>
<td>25</td>
</tr>
<tr>
<td>twister.mpg</td>
<td>9</td>
<td>11065</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 6.1: Frame numbers and average frame size for I, P and B frames

block sizes used in encryption. If we assume that each frame is equally likely to start anywhere in a block and equally likely to end anywhere in a block, on average, we will drop one less block than the number of blocks required to store the dropped frame. If the encryption block size is small when compared with frame sizes, the overhead of this scheme is minimal. Note that in this case, a postprocessing step is needed at the receiver after decryption to remove dropped frame fragments left in the data. Smaller block sizes correspond to encryption algorithms with smaller keyspace and therefore less secure.

Let us first concentrate on some statistical characteristics of the MPEG files used. Table 6.1 provides number of I, P and B frames and the average frame sizes for the MPEG files used. These statistics with the above discussion provides us insight on the expected experimental results.

The next step is to compare how much information we can drop if frame dropping is performed in the decrypted domain with the amount of information we can drop in the encrypted domain. We considered two level video encoding for this purpose. In the first level, all B frames are dropped and in the second level all P and B frames are dropped. We computed the percentage of the amount of data that can be dropped
Figure 6.6: Percentage of data that can be dropped in encrypted form in level 1

in encrypted form compared with data that can be dropped without encryption. We
have considered block sizes of 64, 128, 256 and 512 for encryption. The percentages
for level 1 and level 2 are given in Figures 6.6 and 6.7 respectively.

As expected, as the block size increases, the percentage of data dropped decreases
for levels 1 and 2. In level 1, we can drop more than 95% of data for all files with
encryption block size of 128 bytes. 128 bytes is pretty large for an encryption block,
but we chose this value because we use the light-weight encryption algorithm PC.

The GOP pattern for flower.mpg, bus.mpg and twister.mpg are IBBPBBPBBPBBPBB
and the GOP pattern for tennis.mpg is IBBPBB. In level 1, B frames are dropped,
but fragments of these frames remain in blocks containing I and P frames so the
percentage is lower than it is when compared with no encryption. In level 2, B and
P frames are dropped. There are fewer fragments left in the data since we have fewer
frames in level 2. The size of I frames is also large compared with P and B frames. Together these result in a high percentage of data being dropped.

Twister has the smallest frame sizes on average. This is why the percentage is low compared with other MPEG files for the same block size. Commercial MPEG video services will likely have better quality (larger frames) and the percentages will be even higher. Based on Figures 6.6 and 6.7 frame dropping with strategy 1 is feasible.

**Strategy 2**

In this scheme, the video server uses fillers so that each frame starts at a new block. This allows the proxy to drop all the blocks for a frame to be removed. This scheme increases the size of the stream because of the fillers. Experimental results show that the increase in size due to fillers is reasonably small. The overhead for the
Figure 6.8: Percentage increase in size due to fillers

4 MPEG files versus the percentage increase in size is given in Figure 6.8. As the block size increases the percentage overhead also increases since we need to use more fillers so that the next frame starts in a new block. The filler overhead is less than 5% for bus.mpg, tennis.mpg and flower.mpg and less than 8% for twister.mpg. Strategy 2 is feasible and it is preferable to strategy 1 because of the ease of semantics (frames start on new blocks). Fillers need to be removed at the client side after decoding.

6.3.2 Encryption/Decryption Speed

The proxy will need to do encryption for multiple streams therefore we should use a light-weight encryption algorithm to enable this. The running times of PC for a block of 256 bytes with operations limited to bytes is given in Table 7.3. As shown in the table, the PC algorithm is about 9-10 times faster than gpg. Thus, we can encrypt
<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>PC (sec)</th>
<th>gpg (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus.mpg</td>
<td>718464</td>
<td>0.0793</td>
<td>0.7666</td>
</tr>
<tr>
<td>flower.mpg</td>
<td>719510</td>
<td>0.0848</td>
<td>0.7675</td>
</tr>
<tr>
<td>twister.mpg</td>
<td>401371</td>
<td>0.0452</td>
<td>0.4751</td>
</tr>
<tr>
<td>tennis.mpg</td>
<td>1246001</td>
<td>0.1277</td>
<td>1.2686</td>
</tr>
</tbody>
</table>

Table 6.2: Timings for gpg and PC

![Number of Quality Levels vs Overhead](image)

Figure 6.9: Overhead as a function of quality Levels for Crocodile Dundee

9-10 times as many streams using PC encryption at the proxy. These results were taken using a 400 MHz computer with 256 MB memory running the Linux operating system. Our purpose here is to show the ratio of running times of the two programs. Using a computer with better processing power ratio should be similar.
6.4 Experimental Results on Authentication

Using the proposed authentication scheme, the proxy leaves only a single signed hash with each GOP so the overhead due to multiple hashes is not high between the proxy and client. On the other hand, the server adds multiple signed hashes to each GOP so the overhead between the server and proxy depends on the number of hashes stored with each GOP. For authentication we investigated this overhead for the Crocodile Dundee and E.T movie sequences. The overhead for Crocodile Dundee and E.T. is given in figures 6.9 and 6.10, respectively. When compared with the size of the video data, the overhead is very low. Experimental results show that multiple levels can efficiently be supported for authentication. These results are based on 20-byte hashes.
There is a tradeoff between the security of hash function used and the size of the hash. Hash functions with smaller hash are more vulnerable against birthday attacks because probabilistically it is easier to find 2 messages with same hash. Based on the results of overhead we can use 20-byte hash function SHA for authentication. Although SHA is more computationally expensive than 16-byte hash function MD4, we can use SHA since hashes at the server are precomputed and stored with movie.
CHAPTER 7

WIRELESS VIDEO TRANSMISSION AND SECURITY MECHANISMS

In this chapter, I investigate wireless video transmission and security mechanisms for wireless video transmission. When video is transmitted over a wireless networks, bit errors might be introduced by the channel. Due to the characteristics of the coding used, some of the bit errors will be undetected by the video decoder and some will be detected as error. In this chapter we investigate this in detail. Then I propose a lightweight video encryption framework for wireless video transmission. My framework divides the video into layers and uses different levels of FEC to protect the data. Finally I propose error-preserving encryption for wireless video transmission.

7.1 The Effect of Bit Errors on MPEG Video Quality for Wireless Networks

To study the effect of errors on MPEG video streams, we were most interested in the overall effect of a bit-error on the video stream itself. In particular, we were interested in answering questions like: How much noise does a bit error in the data cause in the resultant video stream?, What percentage of the time will an error in the header or marker cause entire frames to be "lost"?

90
To answer these questions, we first obtained a copy of the flower garden MPEG-1 video stream that is publicly available and has been the subject in many video coding papers. This video stream is 352x240 pixels and consists of 40 frames. Using this video sequence, we first re-encoded the first 4 frames of this video stream to have only 3 frames, one I, P and B frame each. I frame is given in figure 7.1. Each slice in this new encoding consists of one row of macroblocks. The resulting video stream consisted of 217,920 bits (or 28.490 bytes). The MPEG-1 video player we used was the publicly available version distributed by the University of California - Berkeley. We used version 2.0 of this player.

Our experimentation then consisted of perturbing one bit in the video sequence and then playing the resultant video stream through the MPEG player. The goal of doing so was to isolate the effect of a single bit error on the output stream as viewed by the user.
<table>
<thead>
<tr>
<th>Type</th>
<th>Undetected</th>
<th>Undetected %</th>
<th>Bad</th>
<th>Bad %</th>
<th>Error</th>
<th>Error %</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marker</td>
<td>25</td>
<td>13.02</td>
<td>68</td>
<td>35.41</td>
<td>99</td>
<td>51.56</td>
<td>192</td>
</tr>
<tr>
<td>Header</td>
<td>10009</td>
<td>89.29</td>
<td>513</td>
<td>3.69</td>
<td>788</td>
<td>7.03</td>
<td>11210</td>
</tr>
<tr>
<td>Data</td>
<td>203241</td>
<td>93.87</td>
<td>3885</td>
<td>1.80</td>
<td>9392</td>
<td>4.34</td>
<td>216518</td>
</tr>
</tbody>
</table>

Table 7.1: Classification of bits into categories

7.1.1 Playability of Corrupted Video Streams

Using our approach, we classified each video stream (with a bit error in it) into one of three categories:

- **Bad**: The bit-error introduced caused a catastrophic failure in the MPEG player. That is, it exited with under some abnormal condition such as exiting after displaying a message that asks us to verify that it is a valid MPEG file.

- **Error**: The MPEG player reported that an error in the bit stream had occurred but continued to play the video stream (albeit distorted). Thus, the error classification denotes that MPEG player picked up the fact that the stream had an error in it.

- **Undetected**: The MPEG player did not report an error and continued to play the video stream. Under this condition, the viewer may or may not see a distorted image.

Our results are shown in table 7.1. As shown in the table, the marker bits are perhaps the most important bits in the compressed video stream. Nearly 35 % of the bit-errors in marker-bit positions caused catastrophic failure in the MPEG player.
One such example is that the marker is interpreted as a different marker, sending the code of to a different part of the decompression code than was actually required or expected. The data bit errors caused relatively fewer problems to the MPEG player. In particular, 94% of the bit errors went undetected by the player. Somewhat more surprising, we found that 89% of it recovered properly.

### 7.1.2 Effect of Bit Error on Picture Quality

To further study the effect of the errors on the perceived picture presented to the user, we once again perturbed each bit within the video stream one at a time. Using the one-bit perturbed video sequence, we then recorded the luminance pixel difference between the original sequence and the perturbed sequence. We ignored all the streams that caused catastrophic failure. To do this, we saved the original 3 video frames from the flower garden for comparison purposes. We then saved the output of the MPEG player using the perturbed sequences (each sequence with exactly one error in it). Using the original and the decoded data in the perturbed sequences, we did a pixel by pixel difference of the frames and recorded a histogram of pixel-luminance errors.

To give the reader an understanding of the luminance values of the frames. We have plotted the luminance value and the number of pixels with that value in figure 7.2. The luminance values are essentially the grayscale image values. In this graph, we have grouped the differences into groups of 4. Group 0 represents those pixel values that were of value 0, 1, 2, or 3. Group 1 represents those pixel values that were of value 4, 5, 6, and 7. In general, Group $k$ denotes the pixel value was between $4(k - 1)$ and $4k - 1$ including both endpoints. Note that the luminance is an integer from 0 to 255.
The effect that a bit in error has on an image depends on the frame type. This is plotted in figure 7.3 for the 3 types of MPEG frames: I, P and B. This data is taken based on the bits that are classified as undetected. The Berkeley MPEG player displays the remainder of slice as black (luminance value = 0) if it can not decode it. If a pixel is plotted black then it has a high difference with the original. The luminance values of the original which is given in figure 7.2 shows us that the values are biased at some values. For all 3 types of frames as the size of difference increases the number of pixels that differ by that amount decreases. Errors in P frames are less than errors in B frames due to the fact that P frames have a lower compression ratio. A one bit error in a P frame is not as important as a 1 bit error in a B frame. In this experiment, an error in P means that the I frame is correct, and an error in B means that the I and P frames that it is based on are correct. The I frame has more
errors due to the fact that the MPEG player displays the remainder of slice as black if there is an error in the slice.

If there is an error in the I frame, it will cause an error in the P frame and the B frame which are coded based on that frame. We investigated this and found out that there is a slight difference in the quality of the subsequent frames, if there is an error in the I frame. In this experiment we have taken the subsequent frames as correct. So for a P frame, this means that there is an error in the I frame but the P frame is correct, and for a B frame this means that I frame has an error but P and B frames are correct. Errors in a P frame are slightly lower than that in I frame and errors in B frame are slightly lower than P frame. This is because of our assumption that
subsequent frames are received correctly. Still this difference is lower than what we expected.

Similarly if there is an error in the P frame this will cause an error in the image displayed for the B frame. We have done a similar experiment to find out how different it was from the case in which the error was in P frame. The results are given in figure 7.5. The error in P frame is higher than the error in B frame caused by an error in P frame. This is consistent with the effect of errors in I frame on other frames but the difference is higher in this case.

The macroblock sizes for the frames are given in figure 7.6. Black means a smaller size and white means a larger size. The sizes are normalized to 255 for the particular frame.
We calculated the number of pixels changed per bit error in each macroblock with error threshold 8. That is, we considered the errors that are at most 7 pixels away from the original, it can be in either direction. Since the errors drop rapidly, this covers most of the errors. This is given in figure 7.7. A black macroblock indicates more errors and a white macroblock indicates fewer errors. Since the file is encoded one slice per row. It should get lighter towards the end of the slice, because an error in the slice affects the rest of the slice. For macroblocks on the left more pixels can be in error, than the ones on the right, when the player cannot decode. However, the compression ratio of the region and where the macroblock is also affect this. In the I frame there is a region on the left that has higher errors, this is because compression of that region is higher, this can be observed in figure 7.6 which shows the size of each

Figure 7.5: Errors in P frame
Figure 7.6: Macroblock size for I, P and B frames, white means large macroblock, black means small macroblock.

Macroblock. Towards the end of the slices we have more errors, this is because it can potentially lose the start of next slice while it is trying to decode the macroblock.

One of the interesting things is that it seems the end of the frame has fewer errors than the rest of the frame. Actually this is because, when we take data we use the bits that are undetected, for last slice the percentage of bits that are undetected is lower than the rest of the slice. This can be seen in figure 7.8. The black color for
Figure 7.7: Errors for I, P and B frames, white means fewer errors, black means more errors.

the macroblock denotes a lower percentage of undetected bits and white denotes a higher percentage of undetected bits. So for the last slice it probably loses the start of the next frame and it becomes a bad or error bit.
<table>
<thead>
<tr>
<th>Type</th>
<th>Undetected</th>
<th>Undetected %</th>
<th>Bad</th>
<th>Bad %</th>
<th>Error</th>
<th>Error %</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marker</td>
<td>26</td>
<td>13.54</td>
<td>70</td>
<td>36.45</td>
<td>96</td>
<td>50.00</td>
<td>192</td>
</tr>
<tr>
<td>Header</td>
<td>9949</td>
<td>88.75</td>
<td>547</td>
<td>4.88</td>
<td>714</td>
<td>6.36</td>
<td>11210</td>
</tr>
<tr>
<td>Data</td>
<td>200243</td>
<td>92.48</td>
<td>4922</td>
<td>2.27</td>
<td>11353</td>
<td>5.24</td>
<td>216518</td>
</tr>
</tbody>
</table>

Table 7.2: Classification of bits into categories

### 7.1.3 2-bit Errors

Errors in wireless networks typically happen in bursts and not single bits at a time. To get an idea of how things change when there are a burst of errors, we did the same experiment (as the one-bit case), flipping two consecutive bits and identifying the resultant video stream as bad, error or undetected. The results are given in table 7.2. As expected when we have 2 consecutive bit errors, the number of undetected errors decreases. The number of video streams labeled as in error decreases probably because some of the video streams labeled error are now labeled bad when there are 2-bit errors. Somewhat surprisingly the data was not very different from the one bit data. The difference in percentages of bad, error, undetected for marker, header and data are less then 2% compared with that of 1-bit errors.

### 7.2 Light-weight Security Mechanisms for Wireless Video Transmission

In this section, we introduce lightweight video encryption algorithms for transmission over wireless networks. The objective of these encryption algorithms is to reduce the total amount of data encrypted (while providing reasonable privacy and security) and to use multiple FEC codes to reduce the expected number of bit errors
left in the data after decoding. The goal of this work is to provide encryption of video data while increasing the decodeability of the stream in the presence of network bit errors. The first algorithm uses multi-layer video coding (which also helps to increase PSNR) for security and encrypts only a base layer. The second algorithm extends video encryption algorithms to reduce the amount of data encrypted and uses different Bose-Chaudhuri-Hocquenghem (BCH) codes for each level.

Forward Error Correction over noisy channels is a common technique used to decrease losses when data is transmitted. It adds some redundancy to the data so that errors can be detected and corrected. This comes at a cost of the overhead of transmitting extra bits. In real-time video transmission, if a packet is lost, the receiver can request a retransmission but by the time the retransmitted packet arrives, the frame is already displayed so retransmission does not help. FEC can be used in these cases. We use BCH error correction to protect the video stream. BCH is used because it is simple to implement, can correct random bit errors, and used heavily in the literature for video protection.

BCH codes are usually stated with 3 parameters. A BCH(a,b,c) code states that the length of the frame including the data and parity bits is a. The number of information bits of the a total bits is represented by b. The minimum distance between all pairs of code words is c. The parameter c is also a measure of the capabilities of the BCH code. The BCH code has the ability to detect up to c bit errors, or the ability to correct \(\lfloor c/2 \rfloor \) bit errors.
7.2.1 Proposed Encryption Algorithms
Multi-layered Video Encryption

In the MPEG video format a block of pixels has 64 coefficients. Out of these 64 coefficients, the lower order coefficients are the most important because they have more information than other coefficients. This can be observed in figure 7.9. In this figure we plot the number of coefficients in the base and enhancement layers and the average PSNR ratio for a 1000 frame MPEG sequence of *Crocodile Dundee* and *E.T.*

Work on wireless video streaming shows that dividing the video into 2 layers and protecting the base layer with a more powerful FEC increases the PSNR ratio [45]. We can use this observation to build an encryption algorithm. Since the user will not be able to view the enhancement layer if the base layer is lost or not decodable, we need only encrypt the base layer only. There is one point we have to be careful about when using encryption of video for transmission over wireless networks. If video is transmitted and error occurs we would still like to see the video with perhaps some distortion. When we encrypt it, if there is an error during transmission that can not be corrected by FEC then we can not decrypt it. Thus, we should use a powerful FEC for the base layer so that the probability of not being able to decode a frame is very low. The encryption procedure is given in figure 7.10. The video is divided into a base layer and an enhancement layer and only the base layer is encrypted.

**VEA Based Encryption Algorithm**

VEA [39] divides the data into 2 lists based on a key and then XORs these lists. The result of the XOR operation together with encryption of one of the lists is the encryption for the data. On the receiver side, the list is decrypted and XOR operations
are applied again to get the other list. Then the data is reorganized using the key. This algorithm encrypts only half of the data. The algorithm is described in figure 7.11. The Double rectangle denotes that those components compose the encrypted stream. An ellipse denotes that the component is encrypted. In this scheme a single bit error that occurs in XOR result will appear as a single bit error in the decrypted stream.

We propose an extension of that algorithm that decreases the amount of data encrypted to a quarter of the data and can be further extended. The idea is instead of encrypting one of the lists, we can apply the VEA algorithm on that list. So, we divide that list into 2 pieces, XOR the pieces, and encrypt one of the smaller lists. This will result in a quarter of the data being encrypted. Figure 7.12 describes this scheme. This approach can significantly reduce the number of XOR operations, but the amount of data moved between the lists increases. Also in this scheme notice that a single bit error in the first level XOR result causes a single error in the decrypted data but a single bit error in second level XOR result will cause 2 bit errors in the decrypted data. Based on this observation, it is best to use 3 different BCH codes for encrypted sublist, the first level XOR result and the second level XOR result.

Assuming that the encrypted data is transmitted without errors, the expected number of bit errors per frame for a \( N \) level Extended VEA code is:

\[
\sum_{k=1}^{n} (2^k \times E[X_k])
\]  

(7.1)

where \( X_k \) is the \( k \)’th XOR result.
7.2.2 Experimental Results

A Markov channel model of radio channels is described in [70] and used in [54] and many other research works. Two state Markov chains have also been used to model packet loss behavior of the Internet [58], [21]. We model bit errors using a two state Markov chain model for our simulations.

We transmitted 100,000 frames of size 267 bits (including FEC overhead). We used 100 different seeds for the random number generator and transmitted 1000 frames simulating the wireless link using that seed. In the remainder of the section $BCH(i)$ refers to BCH code which has 267 bit codewords and can correct $i$ bit errors. When the frames are decoded, there will be bit errors left in the data due to the use of FEC. For some BCH codes the percentage of frames with bit errors in the data after decoding is given in Figure 7.13. The percentage of frames in error decreases dramatically as better codes are used as can be seen from the figure particularly since encryption does not tolerate errors.

We encrypted some randomly created data using PGP and introduced bit errors into the encrypted data. We observed that PGP was not able to decrypt any of the streams. It gives an error message indicating that the phrase entered for decoding (which substitutes the key) is wrong. Based on this we can conclude that if encrypted data is used in transmission of video data using FEC, the advantages of FEC are completely lost. Video data can tolerate bit errors so using FEC for video is acceptable but only when encryption is not used. Figure 7.13 can be interpreted as the percentage of data that will be lost if encryption is used with FEC.
Figure 7.14 gives the overhead of the BCH code used. As can be seen from the figure, the overhead of FEC is linearly proportional to the number of errors the code can correct.

Since encryption with FEC is intolerable to bit errors, it is possible to increase tolerance by using methods that decrease the amount of data encrypted.

If the total encryption is used with BCH(3). The expected number of bit errors left in a frame after decoding will be 11.07 with an FEC overhead of 10.11%.

If layered encryption is used and the first 5 coefficients are placed in base layer, then the size of base layer is approximately 21% for Crocodile Dundee and 31% for E.T. Using BCH(4) for the base layer and BCH(2) for the enhancement layer will incur 8.15% and 8.82% overhead respectively for Crocodile Dundee and E.T. The expected number of bit errors left in a frame after decoding is 1.1855 for E.T. and 0.9805 for Crocodile Dundee. If VEA is used with BCH(4) and BCH(3) the overall overhead is 11.75% and the expected number of bit errors left in a frame is 1.39. Extended VEA with BCH(4), BCH(3), BCH(2) for 3 levels has 9.26% overhead and the expected number of bit errors left in a frame after decoding is 1.0150. As can be seen from this analysis, layered encryption and the extended VEA algorithm decrease the expected number of bit errors left in a frame significantly for about the same overhead. We expect this decrease to translate to better video quality. Note that this analysis assumes that if there are bit errors in an encrypted frame then all the bits in that frame are lost and considered in error.
7.3 Video Encryption for Transmission over Wireless Networks

In this section, we describe error preserving encryption mechanisms for transmission of video over wireless networks. In previous section, we described schemes for reducing amount of data encrypted so that bit errors do not cause loss of data. We describe error preserving encryption algorithms designed specifically for video to solve the problem of data being lost due to bit errors that hit encrypted fragments. The main objective of this work is to ensure that the basic encryption of the stream can survive bit errors and that the errors are then passed to the application. Experimental results show that the proposed algorithm is secure against ciphertext only attacks, but vulnerable against known plaintext attacks. In ciphertext attack, attacker has only the ciphertext to work with, whereas in known plaintext attacks attacker has the encryption of a known text. Considering the fact that an algorithm which is secure against known plaintext attacks is not suitable for wireless video transmission this is acceptable.

7.3.1 Security Issues Relevant to Wireless Video Transmission

There are several properties that describe good cryptosystems. The Error propagation property [26] requires that cryptograms of close messages should be distributed over the whole cryptogram space. That is, messages that are similar should yield encryptions that are significantly different to prevent known plaintext attacks. A cryptographic transformation has the avalanche effect [26], if whenever a single input bit is complemented, on average one half of the output bits change. The output response of single-bit input changes and double-bit input changes are investigated in [55]. A
cryptographic transformation is complete [35] if each ciphertext bit depends on all the plaintext bits. These properties while inherently good for encryption are bad for wireless video encryption because any bit errors left in encrypted streams after FEC decoding causes large loss of data.

7.3.2 A Theory for Understanding Error Preserving Encryption Functions

As we described previously, there are a number of conflicting goals when transmitting video over wireless networks. We want to be able to provide digital rights management, but we have not parallel limited processing resources and bandwidth, and the use of FEC might render significant amount of data useless if FEC can not correct the bit errors. Using FEC which corrects a very high percentage of bit errors wastes bandwidth since such FEC has high overhead. Consider the following scenario with FEC that can correct 5 bits used for wireless transmission of video. Some bit errors might be left in the data after FEC decoding. In such a situation, the number of bit errors will be 6, 7 or 8 (slightly higher than FEC capability) with high probability. An ideal encryption function would be one which satisfies the condition that if \( E(x) \) and \( E(y) \) differ in \( c \leq 8 \) bits, then \( x \) and \( y \) differ in \( d \leq c \) bits where \( E(x) \) denotes encryption of \( x \). It can be shown that if the encryption function decreases some distances then it must increase some other distances. One class of functions which satisfy the above condition is the class which preserves the distance. We concentrate on encryption functions with the property that if \( E(x) \) and \( E(y) \) differ in \( i \) bits then \( x \) and \( y \) also differ in \( i \) bits. This process is given in Figure 7.15. With errors possibly left after FEC decoding \( E(y) \) is given to the decryption algorithm. Note that \( E(y) \) is \( E(x) \) with errors. If \( E(x) \) and \( E(y) \) differs in \( k \) bits, we want \( x \) and \( y \) to differ
in $k$ bits. Using typical encryption algorithms, $x$ and $y$ will differ in about half the bits. In this section we identify all functions which preserve bit errors and provide methods to generate such functions efficiently for video encryption for transmission over wireless networks.

To produce a framework for how to do this, we will first describe several properties and lemmas that are needed. We then describe how to apply these to video streams. Let $f$ be a function of the form $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ which means that $f$ maps binary strings of length $n$ to binary strings of length $n$. Let $d(x, y)$ be the hamming distance between $x$ and $y$. By definition of hamming distance $d(x, y)$ is the number of bits in which $x$ and $y$ differ. We call a function of the above form error-preserving if $d(x, y) = d(f(x), f(y)) \forall x, y$. In other words if $x$ and $y$ differ in $k$ bits then $f(x)$ and $f(y)$ also differ in $k$ bits. To help visualize such functions we introduce neighbor graphs. Neighbor graphs form a geometrical shape and help understand error preserving functions. We can construct a neighbor graph $NG$ of $\{0, 1\}^n$ by connecting $x$ and $y$ if $d(x, y)=1$. As an example consider binary strings of length 2. There are 4 such strings 00, 01, 10, 11. Since 00 and 01 differ in 1 bit, connect them with an edge. Similarly connect 00 and 10, 01 and 11, 10 and 11. The resulting shape is a square. Similarly construct a neighbor graph $NG(f)$ by connecting $f(x)$ and $f(y)$ if $d(f(x), f(y))=1$. Connect 2 nodes if $f(x)$ and $f(y)$ differ in 1 bit. Neighbor graphs $NG$ and $NG(f)$ form n-dimensional hypercube. For n=2, they are squares, for n=3, they are cubes. A function $f$ is error preserving if it preserves the shape of the hypercube. Error preserving functions map vertices of $NG$ to $NG(f)$ and when doing that shape of the n-dimensional hypercube is preserved. If $x$ and $y$ form an edge in $NG$, then $f(x)$ and $f(y)$ should also form an edge in $NG(f)$. Figure 7.16 gives an example of a function which is error preserving
and Figure 7.17 gives an example of a function which is not error preserving. In Figure 7.17 the function does not preserve the shape of the square. In Figures 7.16 and 7.17 the left part shows $NG$ and right part shown $NG(f)$. In algebra, a transformation that preserve shape is called an isometry. We use the concept of isometry because we use geometry for some of the proofs. The number of error-preserving functions is equal to the number of isometries of an $n$-dimensional hypercube. If a function is error preserving it will be an isometry and if it is not then there must be an edge which violates error preserving property. We refer to readers not familiar with algebra to [27]

**Theoretical Results on Error Preserving Encryption Functions**

In this subsection we prove the following lemmas which form the basis of error preserving encryption. Readers that do not care to read the proofs throughly can move to the following section.

**Lemma 1:** The Number of error preserving functions $\{0, 1\}^n \rightarrow \{0, 1\}^n$ is $n! \times 2^n$

**Proof:** We want to find functions which map an $n$-dimensional hypercube onto another $n$-dimensional hypercube so that bit errors are preserved. Determining the mapping for $0^n$ and its neighbors determines the $n$-dimensional hypercube. There are $2^n$ possibilities for $0^n$ since any given hypercube has $2^n$ vertices. $0^n$ has $n$ neighbors. We can map them in $n!$ ways. Therefore there are $n! \times 2^n$ error preserving functions.

As an example, consider the simplest case where $n = 2$. In this case, the hypercube is a square. We can map 00 to any one of the 4 vertices. Lets say we map 00 to 10 (labeled A in graph). 00 has neighbors 01 and 10. We have 2 choices for mapping its
2 neighbors. Either we map 01 to 00 and 10 to 11 (case I) or we map 01 to 11 and 10 to 00 (case II). It is easier to visualize using Figure 7.18. Depending on how we map the neighbors we get the isometry rotation to left by 90 or reflection along the y axis.

**Lemma 2:** All error-preserving functions can be generated using permutation and complementation. Permutation permutes the positions of bits and complementation complements a subset of the bits.

**Proof:** Permuting the positions of bits is an error-preserving function. Complementing a subset of the bits is also an error-preserving function. Now consider all the functions which can be formed by permutation followed by complementation of a subset of the bits. We can permute the bits in $n!$ ways. We can select a subset of bits in $\{0, 1\}^n$ in $2^n$ different ways. Therefore there are $n! \times 2^n$ functions that can be formed using permutation followed by complementation. To show that all error-preserving functions are generated this way, we need to show that generated functions are unique. Assume an error-preserving function generated by $P_1C_1$ where $P_1$ is a permutation and $C_1$ is a complementation of a subset of bits, assume that the same function is also generated by $P_2C_2$. We have to show $P_1 = P_2$ and $C_1 = C_2$. Clearly if $P_1 = P_2$ then $C_1 = C_2$ and if $C_1 = C_2$ then $P_1 = P_2$. So assume $P_1 \neq P_2$ and $C_1 \neq C_2$. Since $P_1$ is a permutation $P_1$ maps $0^n$ to $0^n$. $P_2$ is also a permutation and $P_2$ maps $0^n$ to $0^n$. Since $P_1C_1 = P_2C_2$, $C_1$ and $C_2$ complement same bits. Therefore $C_1 = C_2$ which implies $P_1 = P_2$. This is a contradiction. So Permutation complementation generates a total of $n! \times 2^n$ error-preserving functions and it generates each error preserving function only once. Therefore it generates all the error-preserving functions using lemma 1.
Lemma 3: \((P_1C_1) \circ (P_2C_2) = (P_3C_3)\) where \(\circ\) denotes function composition.

Proof: Let \(f = (P_1C_1)\) and \(g = (P_2C_2)\). Then \((P_1C_1) \circ (P_2C_2) = f \circ g\). From \((f \circ g)(x) = g(f(x))\) using the fact that \(g\) is error preserving we have \(d(g(f(x)), g(f(y))) = d(f(x), f(y))\). And using the fact that \(f\) is error preserving we have \(d(f(x), f(y)) = d(x, y)\). Therefore \(d(g(f(x)), g(f(y))) = d(x, y)\) and \((P_1C_1) \circ (P_2C_2)\) is error preserving and it can be written as \((P_3C_3)\) by lemma 2.

For simplicity we will not use \(\circ\) for the rest of the section.

Lemma 4: \((P_1C_1)^{-1}(P_2C_2) = (P_3C_3)\)

Proof: For simplicity, in this proof we use the fact that isometries of an \(n\)-dimensional hypercube form a group. Therefore inverses exist. We have \((P_1C_1)(P_2C_2) = (P_2C_3)\) from lemma 3. Multiplying on the left by \((P_1C_1)^{-1}\) we get \((P_2C_2) = (P_1C_1)^{-1}(P_3C_3)\) which is in the form above.

Lemma 5: \((P_1C_1)(P_2C_2)^{-1}(P_3C_3) = (P_4C_4)\)

Proof: Immediate using the fact that isometries form a group, lemma 4 and lemma 3.

Lemma 6: The number of permutations which map none of the bits to their original position is given by \(D_n = n!(1 - 1/(1!) + 1/(2!) - 1/(3!) + \ldots + (-1)^n1/(n!))\).

Proof: Such permutations are called derangements in combinatorics. We refer the readers not familiar to combinatorics to [14].

111
**Lemma 7**: Using the random Derangement-Complementation (D-C) algorithm. The probability that k users will all have distinct (D-C) is

\[ \prod_{j=1}^{k} \left( \frac{D_n 2^n - j}{D_n 2^n} \right) \]

which is larger than

\[ 2^{k \log \left( \frac{D_n 2^n - k}{D_n 2^n} \right)} \]

### 7.3.3 Applications of error preserving encryption functions

In this section, we describe some applications of the lemmas given in the previous section and describe why they are important for multimedia transmission over wireless networks.

Let us first discuss the use of error-preserving functions in encryption. Such functions won’t satisfy the error propagation property, the avalanche effect and the completeness property. These properties mainly guard against known plaintext attacks. That is, an attacker knows a message t, its encrypted version E(t) and is trying to decode another encrypted message E(u). Such an attack on error preserving functions can be done as follows. The attacker finds how many bits differ in E(t) and E(u), let us say a bits differ. He then knows that the plaintext t and u differ in a bits. He can then exhaustively search for those bits. The number of texts which satisfy it are given as combination \( C(n, a) \) where \( n \) is the length of the message.

Considering limited bandwidth, processing resources and power and overhead of FEC to guarantee very low bit-error-rate, we are not in a position to support security at the same level as wired networks. Therefore we should not try to protect against known plaintext attack using some functionality built into encryption. Clearly encryption schemes that satisfy the error propagation property, the avalanche effect
and the completeness property are not suitable for wireless video transmission. Our approach to this problem is as follows. For each video transmission session we use a different (P-C) pair to minimize the chances of known plaintext attack. It is also possible to change the (P-C) pair regularly to guard against such attacks.

Another important issue is why a (P-C) based encryption is suitable for video transmission. Let us first concentrate on the permutation part to motivate the issue. Permutation is generally not a secure encryption method because its use heavily depends on what is being encrypted. Consider a text message encrypted using permutation of size 128 bytes (bytes are permuted). There are 128! ways to order 128 objects. But in text we have some letters that appear many times in a block of size 128 and frequency of vowels is much higher than consonants. There are many such properties of text which makes the attach much easier (most words are 3-8 characters long, words are delimited with space). But video data is compressed and does not exhibit as many hints that can be used for an attack. Theoretically permutation is secure if data sources are stationary and memoryless [28]. Compressed video data exhibits more random behavior than uncompressed video (otherwise there is some correlation and it can be compressed even further. The crucial question at this point is how close compressed video sources are to being stationary and memoryless. To answer this question we used some MPEG files which has been heavily used for research purposes. We investigated the distribution of bytes and distribution of double-bytes. If these are close to uniform then the data source is close to being stationary and memoryless. We are not aware of any tests that measure how close a distribution is to being memoryless and stationary. We used the MPEG files flower.mpg and bus.mpg but the same can be observed with all MPEG files. The byte distribution
for flower and bus video sequences are given in Figures 7.19 and 7.20 respectively. Double-byte distribution for flower and bus video sequences are given in Figures 7.21 and 7.22 respectively. As can be seen from the figures, compressed video data is close to being stationary and memoryless.

Attacking the (P-C) algorithm by generating all the (P-C) pairs is not practical due to the total number of such functions. One possible attack that comes to mind is to create a (P-C) pair and give it to video decoder. As an example, consider a block size of 256 bytes. If the decoder returns an error at byte 1 then eliminate that (P-C) pair and move on to the next one. The idea is to find byte 1, byte 2 and continue this way to find all the bytes in a block. Since the algorithm is block based, once we find the (P-C) pair for a block we are done. This attack strategy does not work because in most cases MPEG player will not fail before byte 2000. To have intuitive idea of why this is the case, consider an arbitrary binary huffman tree and try to decode any string of length $n$. Whenever you reach a leaf, you start decoding from root so you only recognize an error at the end if the process does not end at a leaf node.

The proposed encryption algorithm is as follows. First a connection is established between the video server and client and the (P-C) to be used and initial header and several extra bytes of the MPEG stream are transmitted using some other encryption algorithm such as DES. The reason for this is that the initial segment of the MPEG stream has some fields that can be guessed easily. The size and width of the video for example. The size of this initial video segment will be roughly 160 bytes. Another reason is that for establishing a connection, a standard set of steps needs to be followed and data sent during these steps can be guessed easily if permutation is used because some fixed message type needs to be used.
In the remainder of this section we describe the applications of the lemmas given in previous section.

Lemma 1 gives the expression for the number of error preserving functions. This is important because, an attacker can enumerate all such functions to find the function used. \( n! * 2^n \) is a fast growing function and if \( n \) is chosen properly, such an attack is not feasible. One important point is how can an attacker enumerate only error preserving functions, is there a simple way to do it? We propose a way to enumerate such functions in lemma 2.

Lemma 2 shows that all the error preserving functions can be enumerated using a permutation followed by complementation of a subset of bits. We need a way for doing this so that we can select such a (P-C) pair before transmission. Lemma 2 also gives a practical way to implement such functions using a computer. What we need to do is generate a random permutation, generate a random subset and perform permute and complement. Since operations on bits are costly to manipulate on a computer, for video, we can define these operations on bytes. Implementing permutation can be easily and efficiently done using table look-up. Implementing complementation of bytes is easy since many programming languages offer such operators.

Double encryption [61] is proposed as a way to improve security of a block encryption algorithm. It is based on encrypting twice with 2 different keys. Decryption is the reverse. Lemma 3 shows that for error preserving encryption double encryption does not help since there is a (P-C) that is equivalent to it.

Lemma 4 can be used for secure wireless video multicast in an ad-hoc network where users can be expelled from the group. To accomplish this assume each node forms a group with its children. Construct a multicast tree, and do the multicast as
follows. The root encrypts the video using permutation-complementation (P-C) algorithm and sends it to its subtrees. Each child by Lemma 5 uses the (P-C) equivalent to decrypt its parent’s encryption and reencrypts with its (P-C) and sends it to its children. Note that in this scheme, when a user is expelled only the parent of that user needs to change the (P-C) and send it to its children.

Triple encryption [61] is another way to improve the security of a block encryption algorithm. Here we focus on a version with 3 different keys. Encryption is done by encrypting with first key, decrypting with the second key and encrypting with a third key. Decryption is done by decrypting with the third key, encrypting with the second key and decrypting with the third key. Lemma 5 shows that triple encryption is equivalent to a single encryption and therefore does not improve security.

Some of the permutations such as the ones which keep a majority of the bits in their original positions are vulnerable. We eliminate some of the permutations from the (P-C) algorithm to improve security. We limit ourselves to the permutations which map none of the bits to its original position. Such permutations are called derangements. Lemma 6 gives the expression for number of derangements. The number of derangements converges to \( n!/e \) as \( n \) tends to infinity. So, we still have a large set of error preserving functions to work with. This is good since it makes brute-force attack impractical.

Video-on-demand servers may server hundreds of users, for such systems we want to be able to generate (P-C) pairs with minimum probability of collision. In order to achieve this, for each new user we can randomly select a derangement, a subset of bits for complementation and send it to user. Lemma 7 gives the probability that no collision occurs for \( k \) users. For \( n = 10 \) and \( k = 1000 \) second expression gives
\begin{tabular}{|l|c|c|c|c|}
\hline
File          & Size   & perm. (sec.) & P-C (sec) & gpg (sec.) \\
\hline
bus.mpg       & 718464 & 0.0685      & 0.0793    & 0.7666    \\
flower.mpg    & 719510 & 0.0683      & 0.0848    & 0.7675    \\
twister.mpg   & 401371 & 0.0378      & 0.0452    & 0.4751    \\
tennis.mpg    & 1246001 & 0.1181    & 0.1277    & 1.2686    \\
\hline
\end{tabular}

Table 7.3: Timings for gpg and permutation and P-C

0.99999994520261 which means that probability of collision is very low. For k = 10000 the expression is 0.99999452027058. Note that the second expression in lemma is given for ease of computation, it is a lower bound. The actual value will be higher, and in practice, value of n will be much higher than 10.

The running times of permutation, (P-C) for blocks of 256 bytes with operations limited to bytes is given in Table 7.3. As shown in the table, the permutation based algorithm is about 10 times faster than gpg, and (P-C) algorithm is slightly slower than permutation and about 9-10 times faster than gpg. The algorithm in [39] will be about 2 times faster than gpg since it encrypts about half the data, but it is not an error preserving algorithm. These results were taken using a 400 MHz PC with 256 MB memory running the Linux operating system. Our purpose here is to show the ratio of running times of the two programs. Using a mobile device the ratio should be similar.
Figure 7.8: Percentage of undetected bit-errors for I, P and B frames, white means high percentage, black means small percentage.
Figure 7.9: Average PSNR value for different breakpoints

Figure 7.10: Layered video encryption
Figure 7.11: VEA encryption

Figure 7.12: Extended VEA encryption
Figure 7.13: BCH code vs the number of frames in error for various error rate.

Figure 7.14: BCH code vs percentage of overhead

Figure 7.15: Encryption for wireless video transmission
Figure 7.16: Example of an error preserving function

Figure 7.17: Example of a function which is not error preserving
Figure 7.18: Sample isometries of square

Figure 7.19: Byte distribution for flower.mpg
Figure 7.20: Byte distribution for bus.mpg

Figure 7.21: Double byte distribution for flower.mpg
Figure 7.22: Double byte distribution for bus.mpg
CHAPTER 8

CONCLUSION

With widespread use of multimedia data on the Internet, multimedia security is gaining a lot of interest. However, most of the research on multimedia security have focused on watermarking of images/video and somewhat on video encryption. In this thesis I approached multimedia security from streaming side and investigated security issues in several settings including passive attacks, proxy-based systems, lossless video transmission and wireless video transmission.

Trace of a streaming session depends on movie characteristics as well as available network conditions. Most of the streaming algorithms rely more on movie characteristics and push data over the network using UDP. This creates a problem since movie features are revealed even though the movie is encrypted. I experimentally verified this using time series to represent traces and similarity search to identify the movies. I proposed a framework based on MPEG video coding that reveals no information even if network characteristics are exactly the same. All the movies produces the same trace under the same network conditions. This research has many important consequences since companies will likely add encryption to streaming and present it as secure streaming.
Applications such as medical imaging and scientific visualization demand lossless video compression. The challenge in lossless compression is compression ratio achievable using current systems is low and streaming will be impractical. I proposed a framework that strikes a balance between streaming and lossless display. By dividing the video into two layers, an MPEG layer and lossless differences, and displaying lossless image on demand, bandwidth requirement is reduced for many settings. This framework is easy to secure. We encrypt only the base layer and authenticate the lossless differences. We verified that there is no meaningful information in lossless differences using edge detection methods. Since MPEG layer consists of about 10 % of overall size, encryption costs are significantly reduced.

Many applications involving video streaming use some kind of proxy placed on the network. Such proxies perform adaptation in the form of frame dropping, transcoding and caching. When encryption or authentication is added to the system, things get complicated. I investigated whether end-to-end security can be supported in proxy based systems. End-to-end encryption is feasible using frame dropping and caching. Frames can be dropping from encrypted stream without decoding and caching can be done efficiently on encrypted data. End-to-end authentication is feasible for frame dropping, limited transcoding and caching. By using predetermined quality levels, authentication cost at server and proxy is minimal. The key here is that authentication can be supported using digital signatures. The overhead of authentication was below my expectations. Investigation of security in proxy-based systems is vital since overhead of security will be an important factor that determines the viability of such systems.
Multimedia over wireless networks will be an important part of many applications involving video surveillance for crowd monitoring and video surveillance for environmental protection. In wireless networks due to channel characteristics, some bits will be flipped. I investigated the effect of bit errors on video quality. If bit errors hit encrypted data, then the data is lost. I proposed lightweight schemes for reducing the amount of encrypted data to prevent this loss. I proposed an error-preserving video encryption algorithm for wireless video. The error-preserving algorithm is very efficient, 10 times faster than traditional algorithms and has several desirable properties.

I believe this thesis addresses an important topic on multimedia networking. Some of the ideas presented in this thesis such as passive attacks require further attention. Future work includes interesting problems such as anonymous streaming where server does not know identity of client and client does not know identity of server. Ideas introduced in this thesis can be used to develop a framework for secure collaborative multimedia networking where users get rewarded for helping others.
BIBLIOGRAPHY


[37] F. Korn, N. Sidiropoulos, C. Faloutsos, E Siegel, and Z. Protopapas. Fast and
efficient retrieval of medical tumor shapes. *IEEE Transactions on Data En-
gi neering (TKDE’98).

[38] F. Korn, N. Sidiropoulos, C. Faloutsos, E Siegel, and Z. Protopapas. Fast nearest
neighbor search in medical image databases. In *Proceedings of the Int. Conf. on


[42] C.-S. Li, P.S. Yu, and Castelli V. Hierarchyscan: A hierarchical similarity search

mpeg video, 1999.


Transactions on Circuits and Systems for Video Technology*, 3(3):238–247, June
1993.


In *Proceedings of SPIE, Electronic Imaging, Security and Watermarking of Multi-
tmedia Contents III*.

*IEEE Trans. on Communications*, 44(10), October 1996.


132


[58] Injong Rhee and Srinath R. Joshi. Error recovery for interactive video transmission over the internet. To appear in IEEE journal on selected areas in communications. Special issue on error robust transmission of images and video.


