Acknowledgment

I would like to express my sincere gratitude and appreciation to my supervisor ***Dr. Taha S. Bashaga*** for his guidance, assistance and encouragement through the course of this project.

Special thanks to all my friends, for supporting and giving me advises. And the staff of the department of Computer Science, AL-Nahrain University for their support, interest and generosity.

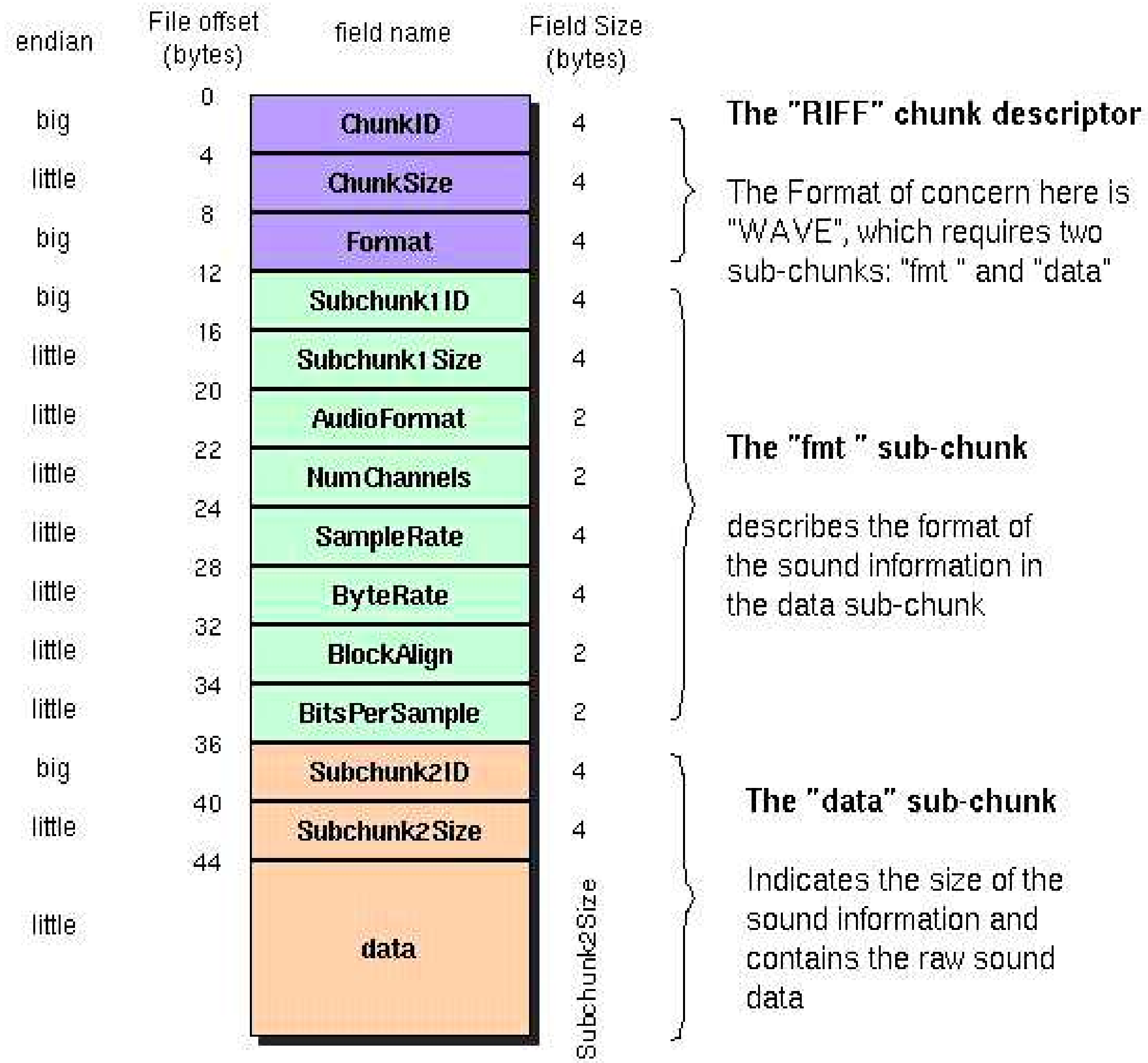
Finally, I want to express my gratitude to my beloved family for encouragement and understanding during the period of my study.

Appendices

Appendix A

The WAVE File Format

The WAVE file format is a subset of Microsoft's RIFF specification for the storage of multimedia files. A RIFF file starts out with a file header followed by a sequence of data chunks. A WAVE file is often just a RIFF file with a single "WAVE" chunk which consists of two sub-chunks -- a "fmt" chunk specifying the data format and a "data" chunk containing the actual sample data. Call this form the "Canonical form".



Offset	Size	Name	Description

The canonical WAVE format starts with the RIFF header:

Offset	Size	Name	Description
0	4	**ChunkID**	Contains the letters "RIFF" in ASCII form (0x52494646 big-endian form).
4	4	**Chunk Size**	36 + SubChunk2Size, or more precisely: 4 + (8 + SubChunk1Size) + (8 + SubChunk2Size) This is the size of the rest of the chunk following this number. This is the size of the entire file in bytes minus 8 bytes for the two fields not included in this count: ChunkID and ChunkSize.
8	4	**Format**	Contains the letters "WAVE" (0x57415645 big-endian form).

The "WAVE" format consists of two subchunks: "fmt" and "data":
The "fmt" sub chunk describes the sound data's format:

Offset	Size	Name	Description
12	4	**Subchunk1ID**	Contains the letters "fmt" (0x666d7420 big-endian form).
16	4	**Subchunk1Size**	16 for PCM. This is the size of the rest of the Subchunk which follows this number.
20	2	**AudioFormat**	PCM = 1 (i.e. Linear quantization) Values other than 1 indicate some form of compression.
22	2	**NumChannels**	Mono = 1, Stereo = 2, etc.
24	4	**SampleRate**	8000, 44100, etc.
28	4	**ByteRate**	== SampleRate * NumChannels * BitsPerSample/8
32	2	**BlockAlign**	== NumChannels * BitsPerSample/8 The number of bytes for one sample including all channels. I wonder what happens when this number isn't an integer?
34	2	**BitsPerSample**	8 bits = 8, 16 bits = 16, etc.
	2	**ExtraParamSize**	if PCM, then doesn't exist
	X	**ExtraParams**	space for extra parameters

The "data" subchunk contains the size of the data and the actual sound:

Offset	Size	Name	Description
36	4	**Subchunk2ID**	Contains the letters "data" (0x64617461 big-endian form).

40 4 **Subchunk2Size** == NumSamples * NumChannels * BitsPerSample/8

This is the number of bytes in the data.
You can also think of this as the size
of the read of the subchunk following this
number.

44 * **Data** The actual sound data.

Notes:

- The default byte ordering assumed for WAVE data files is little-endian. Files written using the big-endian byte ordering scheme have the identifier RIFX instead of RIFF.
- The sample data must end on an even byte boundary. Whatever that means.
- 8-bit samples are stored as unsigned bytes, ranging from 0 to 255. 16-bit samples are stored as 2's-complement signed integers, ranging from -32768 to 32767.
- There may be additional subchunks in a Wave data stream. If so, each will have a char [4] SubChunkID, and unsigned long SubChunkSize, and SubChunkSize amount of data.
- RIFF stands for *Resource Interchange File Format*.

Appendix B

The BMP File Format

The BMP file structure is very simple and is shown in Figure B.1:

File Header	Image Header	Colour Table	Pixel Data

Fig B.1 BMP File Format

B.1 File Header

Every Windows BMP begins with a BITMAPFILEHEADER structure whose layout is shown in Table B.1. The main function of this structure is to serve as the signature that identifies that file format.

Field Name	Size in bytes	Description
bfType	2	Contains the characters "BM" that identify the file type
bfSize	4	File size
bfReserved1	2	Unused
bfReserved2	2	Unused
bfOffBits	4	Offset to start of pixel data

Table B.1 Bit Map file header structure

Three checks can be made to ensure that the file you are reading is in fact a BMP file:

- The first two bytes of the file must contain the ASCII characters "B" followed by "M".
- If you are using a file system where you can determine the exact file size in bytes, you can compare the file size with the value in the bfSize field.
- The bfReservedl and bfReserved2 fields must be zero.

The file header also specifies the location of the pixel data in the file. When decoding a BMP file you must use the bfOffbits field to determine the offset from the beginning of the file to where the pixel data starts. Most applications place the pixel data immediately following the BITMAPINFOHEADER structure or palette, if it is present. However, some applications place filler bytes between these structures and the pixel data so you must use the bfOffbits to determine the number of bytes from the BITMAPFILEHEADER structure to the pixel data.

B.2 Image Header

The image header immediately follows the BITMAPFILEHEADER structure. It comes in two distinct formats, defined by the BITMAPINFOHEADER and BITMAPCOREHEADER structures.

BITMAPCOREHEADER represents the OS/2 BMP format and BITMAPINFOHEADER is the much more common Windows format. Unfortunately, there is no version field in the BMP definitions. The only way to determine the type of image structure used in a particular file is to examine the structure's size field, which is the first 4 bytes of both structure types. The size of the BITMAPCOREHEADER structure is 12 bytes; the size of BITMAPINFOHEADER, at least 40 bytes.

The layout of BITMAPINFOHEADER is shown in Table B.2. This structure gives the dimensions and bit depth of the image and tells if the image is compressed. Windows 95 supports a BMP format that uses an enlarged version of this header. Few applications create BMP files using this format; however; a decoder should be implemented so that it knows that header sizes can be larger than 40 bytes. The image height is an unsigned value. A negative value for the `biHeight` field specifies that the pixel data is ordered from the top down rather than the normal bottom up. Images with a negative `biHeight` value may not be compressed.

Field Name	Size	Description
biSize	4	Header size must be at least 40
biWidth	4	Image width
biHeight	4	Image height
biplanes	2	Must be 1
biBitCount	2	Bits per pixel: 1,4,8,16,24, or 32
biCompression	4	Compression type: BI_RGB=0, BI_RLE8=1, BI_RLE4=2, or BI_BITFIELDS=3
biSizeImage	4	Image size: may be 0 if not compressed
bixPelsPerMeter	4	Preferred resolution in pixels per meter
biyPelsPerMeter	4	Preferred resolution in pixels per meter
biClrUsed	4	Number of entries in the color map that are actually used
biClrImportant	4	Number of significant colors

Table B.2 Bit Map Info Header structure

The BITMAPCOREHEADER structure is the other image header format. Its layout is shown in Table B.3:

Field Name	Size	Description
bcSize	4	Header size must be 12
bcWidth	2	Image width
bcHeight	2	Image height
bcPlanes	2	Must be 1
bcBitCount	2	Bit count: 1,4,8, or 24

Table B.3 Bit Map Core Header structure

Notice that it has fewer fields and that all have analogous fields in the BITMAPINFOHEADER structure. If the file uses BITMAPCOREHEADE-R rather than BITMAPINFOHEADER, the pixel data cannot be compressed.

B.3 Color Palette

The color palette immediately follows the file header and can be in one of three formats. The first two are used to map pixel data to RGB color values when the bit count is 1, 4, or 8 (biBitCount or bcBitCount fields). For BMP files in the Windows format, the palette consists of an array of 2 bitcount RGBQUAD structures (Table B.4). BMP files in OS/2 format use an array of RGBTRIPLE structures (Table B.5).

Field Name	Size	Description
rgbBlue	1	Blue color value
rgbGreen	1	Green color value
rgbRed	1	Red color value
rgbReserved	1	Must be zero

Table B.4 BRGBQUAD structure

Field Name	Size	Description
rgbtBlue	1	Blue color value
rgbtGreen	1	Green color value
rgbtRed	1	Red color value

Table B.5 BRGTRIPLE structure

Appendix C

AVIFile Structures

C.1 Data Structures for AVI Files

Data structures used in the RIFF chunks are defined in the AVIFMT.H header file. The reference section describes the data structures that can be used for the AVI FILE INFO, AVI STREAM INFO. The following structures are used with AVIFile.

C.1.1 AVIFILEINFO

The **AVIFILEINFO** structure contains global information for an entire AVI file.

```
typedef struct {
    DWORD dwMaxBytesPerSec;
    DWORD dwFlags;
    DWORD dwCaps;
    DWORD dwStreams;
    DWORD dwSuggestedBufferSize;
    DWORD dwWidth;
    DWORD dwHeight;
    DWORD dwScale;
    DWORD dwRate;
    DWORD dwLength;
    DWORD dwEditCount;
```

```
    char  szFileType[64];
} AVIFILEINFO;
```

Members:

dwMaxBytesPerSec

Approximate maximum data rate of the AVI file.

dwFlags

Applicable flags. The following flags are defined:

AVIFILEINFO_HASINDEX

The AVI file has an index at the end of the file. For good performance, all AVI files should contain an index.

AVIFILEINFO_MUSTUSEINDEX

The file index contains the playback order for the chunks in the file. Use the index rather than the physical ordering of the chunks when playing back the data. This could be used for creating a list of frames for editing.

AVIFILEINFO_ISINTERLEAVED

The AVI file is interleaved.

AVIFILEINFO_WASCAPTUREFILE

The AVI file is a specially allocated file used for capturing real-time video. Applications should warn the user before writing over a file with this flag set because the user probably defragmented this file.

AVIFILEINFO_COPYRIGHTED

The AVI file contains copyrighted data and software. When this flag is used, software should not permit the data to be duplicated.

dwCaps

Capability flags. The following flags are defined:

AVIFILECAPS_CANREAD

An application can open the AVI file with with the read privilege.

AVIFILECAPS_CANWRITE

An application can open the AVI file with the write privilege.

AVIFILECAPS_ALLKEYFRAMES

Every frame in the AVI file is a key frame.

AVIFILECAPS_NOCOMPRESSION

The AVI file does not use a compression method.

dwStreams

Number of streams in the file. For example, a file with audio and video has at least two streams.

dwSuggestedBufferSize

Suggested buffer size, in bytes, for reading the file. Generally, this size should be large enough to contain the largest chunk in the file. For an interleaved file, this size should be large enough to read an entire record, not just a chunk.

If the buffer size is too small or is set to zero, the playback software will have to reallocate memory during playback, reducing performance.

dwWidth

Width, in pixels, of the AVI file.

dwHeight

Height, in pixels, of the AVI file.

dwScale

Time scale applicable for the entire file. Dividing **dwRate** by **dwScale** gives the number of samples per second.

Any stream can define its own time scale to supersede the file time scale.

dwLength

Length of the AVI file. The units are defined by **dwRate** and **dwScale**.

dwEditCount

Number of streams that have been added to or deleted from the AVI file.

szFileType

Null-terminated string containing descriptive information for the file type.

C.1.2 AVISTREAMINFO

The **AVISTREAMINFO** structure contains information for a single stream.

```
typedef struct {
   DWORD fccType;
   DWORD fccHandler;
   DWORD dwFlags;
   DWORD dwCaps;
   WORD  wPriority;
   WORD  wLanguage;
   DWORD dwScale;
   DWORD dwRate;
   DWORD dwStart;
   DWORD dwLength;
   DWORD dwInitialFrames;
```

```
    DWORD dwSuggestedBufferSize;
    DWORD dwQuality;
    DWORD dwSampleSize;
    RECT  rcFrame;
    DWORD dwEditCount;
    DWORD dwFormatChangeCount;
    char  szName[64];
} AVISTREAMINFO;
```

Members

fccType

Four-character code indicating the stream type. The following constants have been defined for the data commonly found in AVI streams:

Constant	Description
streamtypeAUDIO	Indicates an audio stream.
streamtypeMIDI	Indicates a MIDI stream.
streamtypeTEXT	Indicates a text stream.
streamtypeVIDEO	Indicates a video stream.

fccHandler

Four-character code of the compressor handler that will compress this video stream when it is saved (for example, **mmioFOURCC**('M','S','V','C')). This member is not used for audio streams.

dwFlags

Applicable flags for the stream. The bits in the high-order word of these flags are specific to the type of data contained in the stream. The following flags are defined:

AVISTREAMINFO_DISABLED

Indicates this stream should be rendered when explicitly enabled by the user.

AVISTREAMINFO_FORMATCHANGES

Indicates this video stream contains palette changes. This flag warns the playback software that it will need to animate the palette.

dwCaps

Capability flags; currently unused.

wPriority

Priority of the stream.

wLanguage

Language of the stream.

dwScale

Time scale applicable for the stream. Dividing **dwRate** by **dwScale** gives the playback rate in number of samples per second.

dwRate

See **dwScale**.

dwStart

Sample number of the first frame of the AVI file. The units are defined by **dwRate** and **dwScale**. Normally, this is zero, but it can specify a delay time for a stream that does not start concurrently with the file.

dwLength

Length of this stream. The units are defined by **dwRate** and **dwScale**.

dwInitialFrames

Audio skew. This member specifies how much to skew the audio data ahead of the video frames in interleaved files. Typically, this is about 0.75 seconds.

dwSuggestedBufferSize

Recommended buffer size, in bytes, for the stream. Typically, this member contains a value corresponding to the largest chunk in the stream. Using the correct buffer size makes playback more efficient. Use zero if you do not know the correct buffer size.

dwQuality

Quality indicator of the video data in the stream. Quality is represented as a number between 0 and 10,000. For compressed data, this typically represents the value of the quality parameter passed to the compression software. If set to – 1, drivers use the default quality value.

dwSampleSize

Size, in bytes, of a single data sample. If the value of this member is zero, the samples can vary in size and each data sample (such as a video frame) must be in a separate chunk. A nonzero value indicates that multiple samples of data can be grouped into a single chunk within the file.

rcFrame

Dimensions of the video destination rectangle. The values represent the coordinates of upper left corner, the height, and the width of the rectangle.

dwEditCount

Number of times the stream has been edited. The stream handler maintains this count.

dwFormatChangeCount

Number of times the stream format has changed. The stream handler maintains this count.

szName

Null-terminated string containing a description of the stream.

Appendix D

AVIFile Functions

The following functions are used with AVIFile.

1. AVIFileInit

The **AVIFileInit** function initializes the AVIFile library. The AVIFile library maintains a count of the number of times it is initialized, but not the number of times it was released. Use the **AVIFileExit** function to release the AVIFile library and decrement the reference count. Call **AVIFileInit** before using any other AVIFile functions.

This function supersedes the obsolete **AVIStreamInit** function.

STDAPI_(VOID) AVIFileInit(VOID);

Parameters: This function takes no parameters.

2. AVIFileExit

The **AVIFileExit** function exits the AVIFile library and decrements the reference count for the library.

This function supersedes the obsolete **AVIStreamExit** function.

STDAPI_(VOID) AVIFileExit(VOID);

Parameters: This function takes no parameters.

3. AVIFileInfo

The **AVIFileInfo** function obtains information about an AVI file.

STDAPI AVIFileInfo(

PAVIFILE *pfile***,**

AVIFILEINFO * *pfi***,**

LONG *lSize*

);

Parameters:

pfile

Handle of an open AVI file.

pfi

Address of the structure used to return file information. Typically, this parameter points to an **AVIFILEINFO** structure.

lSize

Size, in bytes, of the structure.

4.AVIFileOpen

The **AVIFileOpen** function opens an AVI file and returns the address of a file interface used to access it. The AVIFile library maintains a count of the number of times a file is opened, but not the number of times it was released. Use the **AVIFileRelease** function to release the file and decrement the count.

STDAPI AVIFileOpen(

PAVIFILE * *ppfile***,**

LPCTSTR *szFile***,**

UINT *mode***,**

CLSID * *pclsidHandler*

);

Parameters:

ppfile

Address to contain the new file interface pointer.

szFile

Null-terminated string containing the name of the file to open.

mode

Access mode to use when opening the file. The default access mode is OF_READ. The following access modes can be specified with **AVIFileOpen**:

OF_CREATE

Creates a new file. If the file already exists, it is truncated to zero length.

OF_SHARE_DENY_NONE

Opens the file nonexclusively. Other processes can open the file with read or write access. **AVIFileOpen** fails if another process has opened the file in compatibility mode.

OF_SHARE_DENY_READ

Opens the file nonexclusively. Other processes can open the file with write access. **AVIFileOpen** fails if another process has opened the file in compatibility mode or has read access to it.

OF_SHARE_DENY_WRITE

Opens the file nonexclusively. Other processes can open the file with read access. **AVIFileOpen** fails if another process has opened the file in compatibility mode or has write access to it.

OF_SHARE_EXCLUSIVE

Opens the file and denies other processes any access to it. **AVIFileOpen** fails if any other process has opened the file.

OF_READ

Opens the file for reading.

OF_READWRITE

Opens the file for reading and writing.

OF_WRITE

Opens the file for writing.

pclsidHandler

Address of a class identifier of the standard or custom handler you want to use. If the value is NULL, the system chooses a handler from the registry based on the file extension or the RIFF type specified in the file.

5. AVIFileRelease

The **AVIFileRelease** function decrements the reference count of an AVI file interface handle and closes the file if the count reaches zero.

This function supersedes the obsolete **AVIFileClose** function.

STDAPI_(ULONG) AVIFileRelease(
PAVIFILE *pfile*
);

Parameters:

pfile

Handle of an open AVI file.

6. AVIFileGetStream

The **AVIFileGetStream** function returns the address of a stream interface that is associated with a specified AVI file.

STDAPI AVIFileGetStream(
PAVIFILE *pfile*,
PAVISTREAM * *ppavi*,
DWORD *fccType*,
LONG *lParam*
);

Parameters:

pfile

Handle of an open AVI file.

ppavi

Address of the new stream interface.

fccType

Four-character code indicating the type of stream to open. Zero indicates any stream can be opened.

lParam

Count of the stream type. Identifies which occurrence of the specified stream type to access.

7. AVIStreamInfo

The **AVIStreamInfo** function obtains stream header information.

STDAPI AVIStreamInfo(
  PAVISTREAM *pavi***,**
  AVISTREAMINFO * *psi***,**
  LONG *lSize*
);

Parameters

pavi

Handle of an open stream.

psi

Address of a structure to contain the stream information.

lSize

Size, in bytes, of the structure used for *psi*.

8. AVIStreamStart

The **AVIStreamStart** function returns the starting sample number for the stream.

STDAPI_(LONG) AVIStreamStart(

PAVISTREAM *pavi*

);

Parameters:

pavi

Handle of an open stream.

9. AVIStreamLength

The **AVIStreamLength** function returns the length of the stream.

STDAPI_(LONG) AVIStreamLength(

PAVISTREAM *pavi*

);

Parameters:

pavi

Handle of an open stream.

10. AVIStreamGetFrameOpen

The **AVIStreamGetFrameOpen** function prepares to decompress video frames from the specified video stream.

STDAPI_(PGETFRAME) AVIStreamGetFrameOpen(

PAVISTREAM *pavi*,

LPBITMAPINFOHEADER *lpbiWanted*

);

Parameters:

pavi

Address of the video stream used as the video source.

lpbiWanted

Address of a structure that defines the desired video format. Specify NULL to use a default format. You can also specify AVIGETFRAMEF_BESTDISPLAYFMT to decode the frames to the best format for your display.

11. AVIStreamGetFrame

The **AVIStreamGetFrame** function returns the address of a decompressed video frame.

STDAPI_(LPVOID) AVIStreamGetFrame(
PGETFRAME *pgf***,**
LONG *lPos*
);

Parameters:

pgf

Address of a **GetFrame** object.

lPos

Position, in samples, within the stream of the desired frame.

12. AVIStreamGetFrameClose

The **AVIStreamGetFrameClose** function releases resources used to decompress video frames.

STDAPI AVIStreamGetFrameClose(
PGETFRAME *pget*
);

Parameters:

pget

Handle returned from the **AVIStreamGetFrameOpen** function. After calling this function, the handle is invalid.

13. AVIFileCreateStream

The **AVIFileCreateStream** function creates a new stream in an existing file and creates an interface to the new stream.

STDAPI AVIFileCreateStream(
  PAVIFILE *pfile***,**
  PAVISTREAM * *ppavi***,**
  AVISTREAMINFO * *psi*
);

Parameters:

pfile

Handle of an open AVI file.

ppavi

Address of the new stream interface.

psi

Address of a structure containing information about the new stream, including the stream type and its sample rate.

14. AVISaveOptions

The **AVISaveOptions** function retrieves the save options for a file and returns them in a buffer.

BOOL AVISaveOptions(
  HWND *hwnd***,**
  UINT *uiFlags***,**

int *nStreams***,**

PAVISTREAM * *ppavi***,**

LPAVICOMPRESSOPTIONS * *plpOptions*

);

Parameters:

hwnd

Handle of the parent window for the Compression Options dialog box.

uiFlags

Flags for displaying the Compression Options dialog box. The following flags are defined:

ICMF_CHOOSE_KEYFRAME

Displays a Key Frame Every dialog box for the video options.

ICMF_CHOOSE_DATARATE

Displays a Data Rate dialog box for the video options.

ICMF_CHOOSE_PREVIEW

Displays a Preview button for the video options. This button previews the compression by using a frame from the stream.

nStreams

Number of streams that have their options set by the dialog box.

ppavi

Address of an array of stream interface pointers. The *nStreams* parameter indicates the number of pointers in the array.

plpOptions

Address of an array of pointers to **AVICOMPRESSOPTIONS** structures. These structures hold the compression options set by the dialog box. The *nStreams* parameter indicates the number of pointers in the array.

15. AVISaveOptionsFree

The **AVISaveOptionsFree** function frees the resources allocated by the **AVISaveOptions** function.

LONG AVISaveOptionsFree(
  int *nStreams*,
  LPAVICOMPRESSOPTIONS **plpOptions*
);

Parameters:

nStreams

Count of the **AVICOMPRESSOPTIONS** structures referenced in *plpOptions*.

plpOptions

Address of an array of pointers to **AVICOMPRESSOPTIONS** structures. These structures hold the compression options set by the dialog box. The resources allocated by **AVISaveOptions** for each of these structures will be freed.

16. AVIStreamSetFormat

The **AVIStreamSetFormat** function sets the format of a stream at the specified position.

STDAPI AVIStreamSetFormat(
  PAVISTREAM *pavi*,
  LONG *lPos*,
  LPVOID *lpFormat*,
  LONG *cbFormat*
);

Parameters:

pavi

Handle of an open stream.

lPos

Position in the stream to receive the format.

lpFormat

Address of a structure containing the new format.

cbFormat

Size, in bytes, of the block of memory referenced by *lpFormat*.

17. AVIStreamWrite

The **AVIStreamWrite** function writes data to a stream.

STDAPI AVIStreamWrite(
PAVISTREAM *pavi***,**
LONG *lStart***,**
LONG *lSamples***,**
LPVOID *lpBuffer***,**
LONG *cbBuffer***,**
DWORD *dwFlags***,**
LONG * *plSampWritten***,**
LONG * *plBytesWritten*
);

Parameters

pavi

Handle of an open stream.

lStart

First sample to write.

lSamples

Number of samples to write.

lpBuffer

Address of a buffer containing the data to write.

cbBuffer

Size of the buffer referenced by *lpBuffer*.

dwFlags

Flag associated with this data. The following flag is defined:

AVIIF_KEYFRAME

Indicates this data does not rely on preceding data in the file.

plSampWritten

Address to contain the number of samples written. This can be set to NULL.

plBytesWritten

Address to contain the number of bytes written. This can be set to NULL.

Chapter One

Introduction

Chapter Two

Steganography Concept

Chapter Three

System Design and Implementation

Chapter Four

Conclusions and Future Work

CHAPTER FOUR

Conclusions and Future Work

4.1 Conclusions

From the test results conducted on the proposed system, the following remarks were derived:

1. Even if the AVI file is very small as video audio file. It is obvious that the video part consist of a very large number of frames even in this small such file. One can hide a reasonable large amount of information, by spreading this information on the video and audio parts.

2. It is obvious that the transmission time increased according to the increasing the size of the file, and the security increased in increasing the cover size to make a compromise between these two factors, and depend on the application.

3. The security of hided information also depends on the method of hiding, for example Haar Wavelet Transform is more secure than Least Significant Bit. But in both cases and since the size of cover is too large the attacking is very difficult.

4. The output stego file remains of same size as the original file. It is also dose not effected after hiding the information according to the subjective measures (seeing and hearing) or the objective measures (MSE and PSNR) as it appears in the results given in chapter three.

4.2 Future Work

During the development of the proposed system, many suggestions for future work was emerged to increase the system efficiency, among these suggestions are following:

1. Develop system that used another hiding video file format, image file format or audio file format.
2. Develop system for hiding video in video by used the same technique used in proposed system or other techniques.
3. Develop system to make a special program used to split audio from video.
4. Develop system that used other hiding methods like (DCT, spread spectrum, etc).
5. Develop system to encrypt the secret data before the hiding process; this will lead to more immunity against extracting attacks.

CHAPTER ONE

Introduction

Security system is one of the most important subjects that take a wide range of importance in many fields, one of which is computer field. In computer system, security is an important issue for data protection. A more powerful and secure systems were needed to protect the transmitted data from being attacked by intruders. Information hiding is one of the powerful secure mechanisms [Sed01]. Steganography is the art and science of hiding information such that its presence cannot be detected [Cac98].

Through the history, the word Steganography comes from the Greek *steganos* (covered or secret) and *graphy* (writing or drawing) and means, literally, covered writing and multitude of methods and variations have been used to hide information [Wat01].

The onset of computer technology and the Internet has given new life to steganography and the creative methods with which it is employed. Computer-based steganography techniques introduce changes to digital carriers to embed information foreign to the native carriers. Since 1995, interest in steganography methods and tools as applied to digital media has exploded [Joh01].

With the change in technology, steganography has transformed into modern steganography. Modern steganography is the ability to hide information in an electronic source of data that appears to be above suspicion to the naked eye [Dor03].

1.1 Some Application of Information Hiding

There are a number of applications driving interest in the subject of information hiding: [Kat00] [Pet99]

- Military and intelligence agencies require unobtrusive communications. Even if the content is encrypted, the detection of a signal on a modern battleeld may lead rapidly to an attack on the signaller
- Criminals also place great value on unobtrusive communications. Their preferred technologies include prepaid mobile phones, mobile phones which have been modified to change their identity frequently, and hacked corporate switchboards through which calls can be rerouted.
- Law enforcement and counter intelligence agencies are interested in understanding these technologies and their weaknesses, so as to detect and trace hidden messages.
- Recent attempts by some governments to limit online free speech and the civilian use of cryptography have spurred people concerned about liberties to develop techniques for anonymous communications on the net.
- Schemes for digital elections and digital cash make use of anonymous communication techniques.
- Marketers use email forgery techniques to send out huge numbers of unsolicited messages while avoiding responses from angry users.

1.2 Literature Survey

Several researches in the information-hiding field tried to insert an additional robust and invisible data into different digital media to achieve a specific type of protection, this survey is limited to the researches that are concerned with steganography. Some of these are summarized below:

- A. Westfeld and G. Wolf (1998) described a steganographic system, which embeds secret messages into a video stream. Examine the signal path, which typically includes discrete cosine transformation (DCT) based, lossy compression. Result is the technical realization of a steganographic algorithm whose security is established by indeterminism within the signal path.
- J. J. Chae and B. S. Manjunath (1999) propose a video data embedding scheme in which the embedded signature data is reconstructed without knowing the original host video. The proposed method enables high rate of data embedding and is robust to motion compensated coding, such as MPEG- 2. Embedding is based on texture masking and utilizes a multi- dimensional lattice structure for encoding signature information. Signature data is embedded in individual video frames using the block DCT. The embedded frames are then MPEG- 2 coded. At the receiver, both the host and signature images are recovered from the embedded bit stream.
- F. Adel (2000) used the two LSBs insertion in many ways (in selecting the embedding positions) such as hopping, sequencing, and hiding in edges of the images. Cryptography also used to encipher the message before it is being embedded in the cover image. A good capacity, approximately quarter image size was obtained.

- C. John (2003) an article about hiding text in the bitmap frames of uncompressed AVI files. This article is about extracting these bitmaps and re-building the stream, in order to hide a message in the video. To hide a message, open a bitmap file, then enter a password or select a key file. This password or key will be treated as a stream of bytes specifying the space between two changed pixels. When enter the secret message, the application writes the length of the message in bytes into the first pixel. After that it reads a byte from the message, reads another byte from the key, and calculates the coordinates of the pixel to use for the message-byte. It increments or resets the color component index, to switch between the R, G and B components. Then it replaces the R, G or B component of the pixel (according to the color component index) with the message-byte, and repeats the procedure with the next byte of the message.
- S. N. Merchant, A. Harchandani, S. Dua, H. Donde, I. Sunesara (2003) This paper proposes a novel video watermarking technique to embed a digital watermark in video data using integer- to- integer discrete wavelet transform. The watermark is embedded in the lowest frequency components of each frame. The method exploits features of the human visual system. The watermark can be extracted directly from the decoded video without access to the original video. Experimental results have indicated that the method is robust against mpeg encoding and re- encoding. It is also perceived that the method is effective against statistical attacks.
- H. J. Mohammed (2004) used the wavelet transform with sorting to embed the image in the cover to increase stego-image robustness against attackers. The embedded image is reduced before hiding into

the cover in order to increase the capacity of the stego system. The redection used is wavelet-based technique with thresholing. To increase system security, wavelet packet algorithm is also used. The random sorting of the subbands prior to embedded is used so that the sorting order is sent as a message-driven key.

- M. J. Jawad (2005) used an audio in audio steganography system, in the proposed system the secret data in the first stage transformed using wavelet transform and then the result and coefficient have to be coded using one of the three coding methods (fixed length encoding method, S-shift coding method and hybrid coding method). In the next stage the embedded stage where the output of coding data (a stream of bits) is embedded in the cover data. Three embedding methods were implemented in this proposed system (least significant bit insertion in wavelet transform domain, two least bits insertion in time domain with recovery technique and hiding in audible part).

1.3 Research Objectives

This work aims to design and implement digital AVI steganography through inserting additional information into the original digital audio\video. An audio\video data embedding scheme in which the embedded signature data is reconstructed without knowing the original host audio\video. The proposed method enables high rate of data embedding and is robust to motion compensated coding. In this work three types of embedding data were used (text, audio and image). This data is embedded in individual video frames using two hiding methods; they are:

- **Spatial Domain Embedding:** One of the widely known steganography algorithms is based on modifying the least significant bit layer of video frames or audio, hence known as the LSB technique. This technique makes use of the fact that the least significant bits could be changed without causing a significant effect on the video frames as well as audio. Although the image seems unchanged visually after the LSBs are modified, the statistical properties of the video frames as well as audio changes significantly.
- **Transform Domain Embedding:** Another category for embedding techniques for which a number of algorithms have been proposed is the transform domain-embedding category. The techniques that are currently being used with video frames can be generalized for use with wavelet transforms

1.5 Thesis Layout

The follows are the outline of the thesis contents:

Chapter One: Includes an introduction to information hiding and literature review to the related steganography work.

Chapter Two: Introduces a background to information hiding, methods of steganography, wavelet transformation with its types, uses and benefits.

Chapter Three: Presents the proposed system design steps, and the practical work implementing these steps. Each practical step is discussed with its related algorithm. Explores and discusses the experimental results of each steganography approach

Chapter Four: Gives some concluding remarks and suggestions for future work.

.

CHAPTER THREE

System Design and Implementation

3.1 Introduction

The main idea of the steganography that described in chapter two is the art of hiding secret message in a host media. Several types of media could be used as host media for hiding secret message. The multimedia steganography (in which the host media is an AVI file) is the aim of this project.

This chapter concerned with the description of the design and implementation part of this project. The description will include how treat with the AVI file. Also, for the steganography methods will be discussed. Finally, since for each steganography technique an appropriate extraction technique is needed, therefore the implemented extraction will also be described.

3.2 The Overall System Model

The overall system model can be described as shown in Fig. 3.1 into five main part as follows: -

- ***Input cover and message files***
- ***Hiding information***
- ***Stego file***
- ***Extraction***
- ***Secret file***

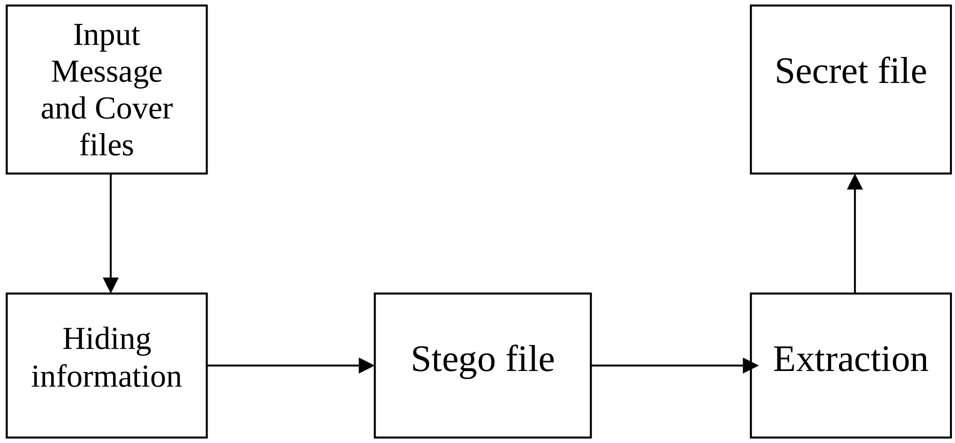


Fig. 3.1 The Overall System Model

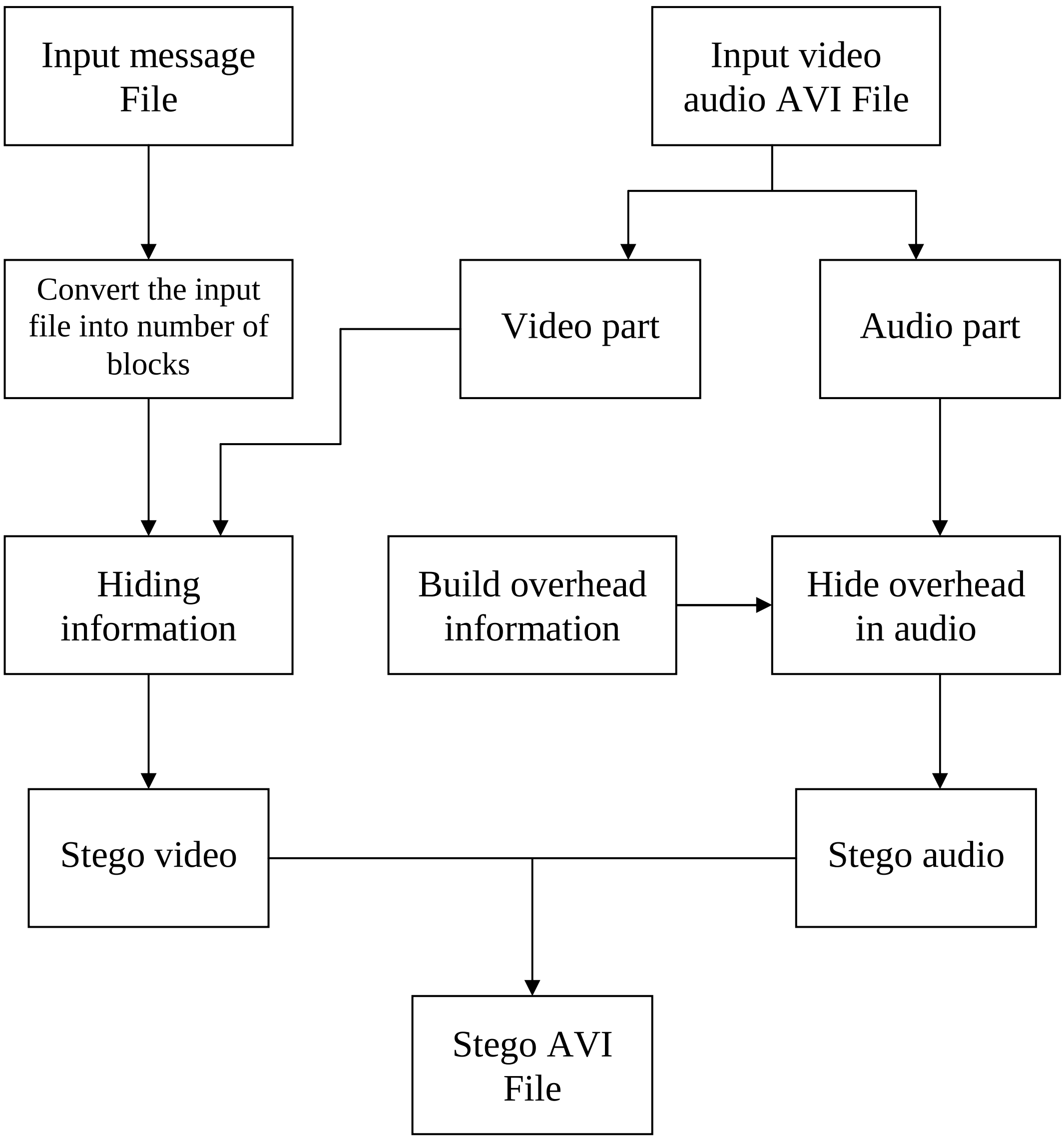


Fig. 3.2 Video audio information hiding

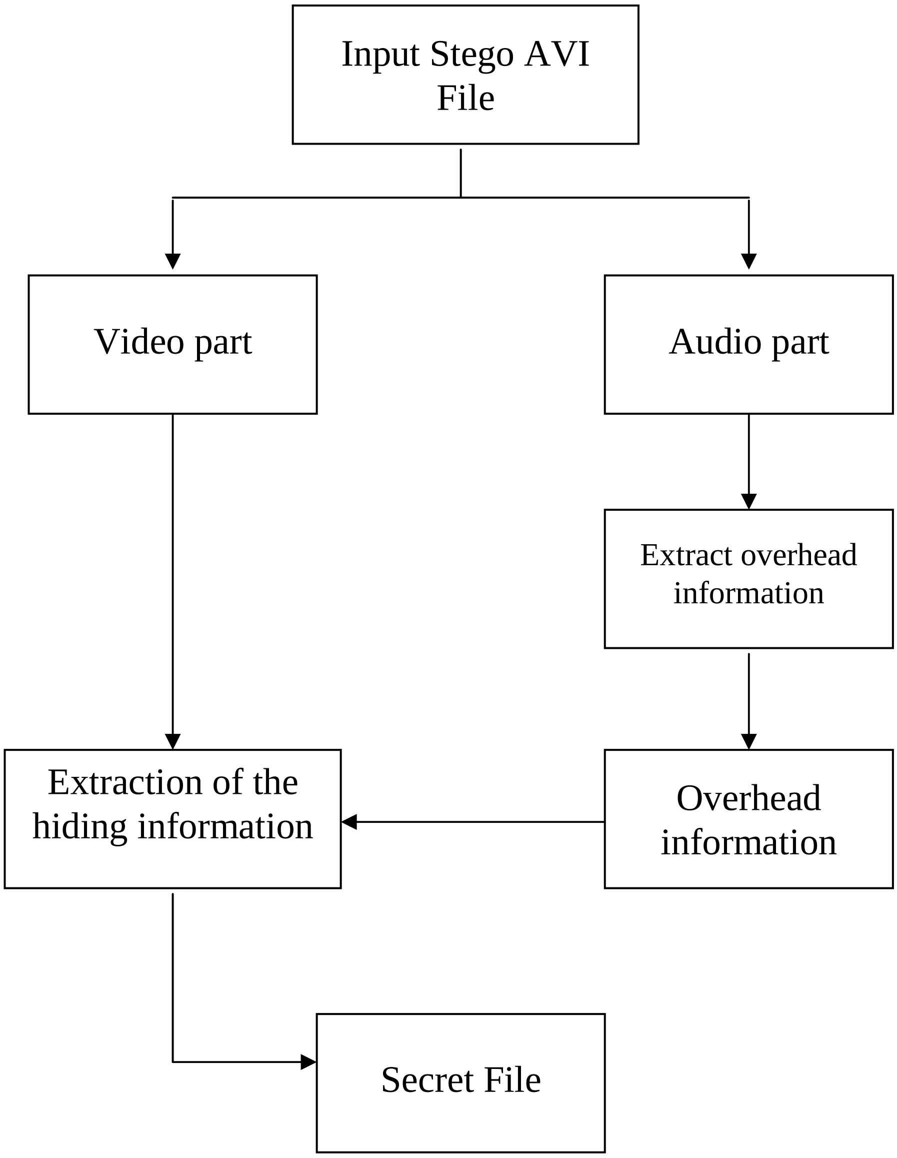


Fig. 3.3 Extraction

3.2.1 Input cover and message files

The actual process of the steganography usually involves two classes of files- cover files and message files.

A. Cover files:

The type of cover file is an AVI file stands for Audio Video Interleave. The AVI file contains the video and audio streams. The main idea is to separate audio part and video part into two separates files and hides deferent information in each of them. This process described by:

- **The Video part** stored on separate file as uncompressed stream of frames of images only (without the audio), the characteristics of the images in the video file are describe to as 32 bits per pixel format has the most significant byte of the pixel set to zero. Then 8 bits for red, 8 bits for green, 8 bits for blue, and 8 bits for reserved. This is RGB8 32 bits. The video stream is divided into frames each stored in separation file as bitmap file, this process done by algorithm (3.1).

Algorithm 3.1 Divided frames into separated BMP files

```
INPUT
        AVI file or AVI video file

OUTPUT
        Save no. of BMP file in special directory
        Firstframe   // Position of the first video frame
        Numframes // Number of frames in video stream
Begin
Open AVI file for read
Read AVI header
Set Firstframe = start stream from header
Set Numframe = length stream from header
Loop For i = Firstframe To (Numframes - 1) + Firstframe
  Read frame (i)
  Convert frame (i) to BMP image file
  Store BMP image file in temporary folder
End Loop
End
```

- **The Audio part** of the Microsoft AVI is separated as WAV (Windows Audio Visual) type format. After the separation of the audio it is stored in WAV file often contains single "WAVE" chunk, which consists of two subchunks, a format (fmt) subchunk specifies the file format and the (data) subchunk containing the actual data samples. Algorithm (3.2) used to open WAV file.

Algorithm 3.2 Open WAV file

```
INPUT
        WAV file
OUTPUT
        X () // Array of one dimension
        Szfile //  size of array
Begin
Open WAV file for read
Read WAV header
Read Szfile
Read the data byte by byte and stored them in array X
End
```

B. Message file:

In the project used unlimited message size can be hidden in the cover. Three types of messages can be used, there are:

I. Text file:

The technique of the text file is a variation of characters. When to be hide in the cover must convert character value (ASCII code) to binary code. Algorithm (3.3) presents the steps of open TXT file.

Algorithm 3.3 Open Text file

```
INPUT
        Text file
OUTPUT
        X () // Array of one dimension
        Szfile //  size of array
Begin
Open Text file for read
 Read Szfile
 Read the data byte by byte and stored them in array X
End
```

II. Image file:

The secret image is considered file of type BMP. The BMP starts out with header followed by a sequence of byte. The size of BMP header is 54 bytes and the data of the image is beginning from the byte 55 to the end of the image size. The characteristics of the image is described as 24 per pixel, each pixel have three bands (Red, Green and Blue) each band size is one byte. To hide this data in the cover each byte of the data must converted to binary code. Algorithm (3.4) presents the steps of open BMP file.

III. Audio file:

The considered secret audio files are of type WAV. The WAV starts out with header followed by a sequence of bytes. The WAV header is 44 bytes and the data of the audio is beginning from the byte 45 to the end of the audio size. The data sample is 8 bits and mono channel. For hiding this data in the cover each byte must convert to binary code. Algorithm (3.2) presents the steps of open WAV file.

The array of secret bytes that got from the three previous algorithms must converted to binary ASCII code by using in algorithm (3.5).

Algorithm 3.4 Open BMP file

```
INPUT
      BMP file

OUTPUT
      X () // Array of one dimension
    Szfile //  size of array
Begin
Open BMP file for read
Read BMP header
Read Szfile
Read the data byte by byte and stored them in array X
End
```

3.2.2 Hiding information

This section will describe the hiding of secret message into AVI cover; it is done in three steps:

A. Preparation step: Before implementing the hiding process a number of steps, are needed to be done:

- Choose the method of hiding in the video stream.

Algorithm 3.5 converting the secret message into binary ASCII code

```
INPUT
        X () // Array of one dimension represent the secret data
        N   // number of characters
OUTPUT
        B () // Array of one dimension represent the binary code of data
        IB  // the size of B array
Begin
Set IB=0
Loop For i=0 to N-1
  Set bits = 1
  Loop For k=0 to 7
    If (X (i) AND bits) Then arrbit (IB) = 0 Else arrbits (IB) = 1
    Set bits = bits*2
    Set IB = IB +1
  End Loop
End Loop
End
```

- Compute the minimum number of frames needed to be used for hiding the given secret messages.
- Choose the number of frames to be used as a cover such that it is greater than or equal the minimum number of frames computed in previous step. Algorithm (3.6) illustrates all these steps.
- Convert the sequential counter of the frame index randomly by using a random Key generation it is amplitude is the number of available frames. Algorithm (3.7) illustrated the random generation.

Algorithm 3.6 Choose the number of frames

```
INPUT
        Mtype // Method types used in project
        Szdata// The size of secret binary code data
OUTPUT
        Noimg // Number of images used as cover
Begin
If Mtype=0 then   // LSB method
Szimg= W * H
If Mtype =1 then   // Wavelet method
Szimg = (W/2) * (H/2)
Minimg= Szdata / Szimg
Input Noimg limited from Minimg to Numframe
End
```

- Divided the message data into number of block that this division suitable to the number of video frames. Each block will be hidden in one frame.

B. Video step: Hiding secret data into the video stream by using two methods mentioned in chapter two:

I. **Least Significant Bit Embedding:** This hiding approach is described in section 2.3.2.A, it is one of the basic and easily implemented image steganography approaches, this is done by hiding one bit from the secret ASCII code data into one pixel of the cover, the given bit hidden at the blue byte of this pixel. This is presented in algorithm (3.8).

Algorithm 3.7 Random Generation

```
INPUT
        Key           // Key number (obtain from algorithm 3.6)
       Numframes // Number of frames in video stream
OUTPUT
        R()    // Arry of index as random
Begin
Loop i=1 to Tindex
  R(i) = i
End Loop
Loop i = Tindex down to 1
   j = i * Rnd
  Swap (R (i), R (j))
End Loop
End
```

Algorithm 3.8 Least Significant Bit Embedded

```
INPUT
        Framedata() //  data of frame cover
        Blkdata()    //  block of message data
        szblk         //  size of block
OUTPUT
        Framedata()    // data of stego image
Begin
Loop s=1 to szblk
Framedata(s).blue = Framedata(s).blue And  254
Framedata(s).blue = Framedata(s).blue Or Blkdata(s)
End Loop
End
```

II. **Haar Wavelet Transform:** The Haar transform is the simplest wavelet transforms, but even this simple method illustrates the power of the wavelet transform. This method is described in section 2.4.2 and hiding is done on the blue bytes of each frame.

The method computes the wavelet transform of the image by alternating between rows and columns. The first step is to calculate averages and differences for all the rows (just one iteration, not the entire wavelet transform). This creates averages in the left half of the image and differences in the right half. The second step is to calculate averages and differences for all the columns, which result in averages in the top-left quadrant of the image and differences in elsewhere. This process obtains from section 2.4.2 and illustrated in algorithm (3.10).

Algorithm 3.9 Haar Wavelet Transform

```
INPUT
        Realdata() //  data of frame cover  as real type
        H              //  Height of frame
        W            //  Width of frame
OUTPUT
        Realdata()    // data of wavelet
Begin
Loop for r =1 to W
  Algorithm (3.10)  Call WTstep (row r of Realdata, W)
End loop
Loop for c = H to 1
   Algorithm (3.10) Call WTstep (column c of Realdata, H)
End loop
End
```

Algorithm 3.10 WTstep

```
INPUT
        Datavector() //  vector of data refers as row or column
        Szvector     //   size of vector
OUTPUT
        Datavector()// data of wavelet
Begin
Loop k =1 to Szvector/2
   DV(k) = (Datavector(2*k-1) + Datavector(2*k))/√2
   DV(Szvector/2+k) = (Datavector(2*k-1) - Datavector(2*k))/√2
End Loop
Datavector = DV
End
```

The result of the algorithm (3.10) is four subbands, The most interesting is the upper left subband, denoted by LL. Then the secret data will be hidden in the upper left subband, that hiding one bit into one coefficient by using a hiding mechanism, that changed the secret bit and summation the result with the coefficient. This is illustrated in algorithm (3.11).

After done the algorithm (3.11) returned the subbands to the original frame this process obtain also from section 2.4.2. This process illustrated in algorithm (3.13).

Algorithm 3.11 Hide secret data in LL subband

```
INPUT
        LL()       //  data of upper left subband
        Blkdata() //  block of message data
        szblk      //  size of block

 OUTPUT
        LL()     //  data of  upper left subband

Begin
Set Step=8
Loop s=1 to szblk
  If Blkdata(s) = 0 then  Blkdata(s)= - factor
  If Blkdata(s) = 1 then Blkdata(s) = factor
  If LL(s)<0 thensign = -1 Else sign = 1
  LL(s) = ABS (LL(s))
  LL(s) = step* round ( LL(s) / step )
  LL(s) = LL(s)+ Blkdata(s)
  LL(s) = sign*LL(s)
End Loop
End
```

Algorithm 3.12 Haar Wavelet Reconstruct

```
INPUT
        Realdata() //  data of frame cover  as real type
        H             //  Height of frame
        W            //  Width of frame
OUTPUT
        Realdata()// data of wavelet

Begin
Loop for c=H to 1
   Algorithm (3.13) Call WRrstep (column c of Realdata, H)
End loop
Loop for r=1 to W
  Algorithm (3.13) Call WRstep (row r of Realdata, W)
End loop
End
```

Algorithm 3.13 WRstep

```
INPUT
        Datavector() //  vector of data refers as row or column
        Szvector     //   size of vector
OUTPUT
        Datavector()// data of wavelet

Begin
Loop k=1 to Szvector/2
  DV(2*k-1) = (Datavector (k) + Datavector (Szvector /2+k))/√2
  DV(2*k) = (Datavector (k) - Datavector (Szvector /2+k))/√2
End Loop
Datavector = DV
End
```

C. Audio step: The third step in the hiding process is to hide the overhead information. The overhead information is hidden in the fixed way in an audio of the AVI file (by using Least Significant Bit). The construction of the overhead information by using fixed number of bits as following:

1 bit Referred to the type of hiding methods

2 bits Referred to the type of message files

32 bits Referred to the number of frames used as cover

If the message file is text:

32 bits Referred to the size of file

If the message file is image:

32 bits Referred to the width of image file

32 bits Referred to the height of image file

If the message file is audio:

32 bits Referred to the size of file

2 bits Referred to the samples rate

Algorithm 3.14 Least Significant Bit Embedded in audio

```
INPUT
        audiodata()  //  data of audio cover
        overh()      //  Overhead data
        szh          //  size of the overhead data
OUTPUT
        audiodata()// data of stego audio

Begin
Loop s=1 to szh
audiodata(s) = audiodata(s) And  254
audiodata(s) = audiodata(s) Or overh(s)
End Loop
End
```

3.2.3 Stego File

After implementing all steps of hiding, the next process is to rebuilt the video stream from the sequence of BMP images by using AVI functions that present in VB6. This process is illustrated in algorithm (3.15). At the beginning, the program builds the header of the video stream and put it in the stego-video. Then, it put the header of each frame and the data one by one. After embedding the last frame the application closes both video files, deletes the temporary bitmap file.

After this step the video stream should merged with audio stream to construct the stego AVI file.

Algorithm 3.15 Construct the Video stream

```
INPUT
        Number of BMP file in special directory
        Firstframe   // Position of the first video frame
        Numframes // Number of frames in video stream
OUTPUT
        Fname of stego video file

Begin
Open Fname for write
Built AVI header
Built frame header
Loop For i = Firstframe To (Numframes - 1) + Firstframe
   Write header in video file
   Write the data of BMP image file as data of frame in video
    Delete the BMP image file from temporary folder
End Loop
End
```

3.2.4 Extraction

It is the art of extracting the hidden data embedded in the AVI carrier file. To accomplish this task, the user who receives the AVI file must divided it into Video and Audio, and extract the overhead information from audio file to arrived to the positions of the secret message. That can illustrate in algorithm (3.16).

From the overhead can determine the type of message, type of method and number of frames to be used as cover. From the number of frames can get the number of block and the size of each block. The number of frames also represents the first frame cover and from it can get the next frame and so on.

The video file will open by used the algorithm (3.1). and the audio file opened by the algorithm (3.2).

Algorithm 3.16 Least Significant Bit Extraction From Audio

```
INPUT
        audiodata()  //  data of audio cover
        szh          //  size of the overhead data

OUTPUT
        overh()      //  Overhead data

Begin
Loop s=1 to szh
  overh(s) = audiodata(s) And  1
End Loop
End
```

Since every method has its own corresponding method to extract secret data from its host video file:

I. Least Significant Bit Extraction: in this method the data extracted from the LS2Bs from frame data portion bytes. An extraction method was build to extract the data sequentially being embedded in the host video. The number of bits extracted from byte of frame data portion depends on the number of bits being embedded in this byte. This process illustrate in algorithm (3.17).

Algorithm 3.17 Least Significant Bit Extraction

```
INPUT
        Framedata() //  data of frame cover
        szblk        //  size of block

OUTPUT
        Blkdata()    //  block of message data

Begin
Loop s=1 to szblk
  Blkdata(s) = Framedata(s).blue And 1
End Loop
End
```

II. Haar Wavelet Reconstruction: This extraction method was done in two steps:

- Used the algorithm (3.10) to convert each frame into four subband.

- Extract the secret data from the upper left subband by using the extraction module, that make round to each coefficient and get different between the coefficient and its rounded, and then check the difference if it is greater than zero then the bit refers to 1 otherwise refers to zero. This process as illustrated in algorithm (3.18). The number of bits extracted from subband depends on the number of bits being embedded in this subband.

Algorithm 3.18 Extracted secret data from LL subband

```
INPUT
        LL()        //  data of upper left subband
        szblk       //  size of block

OUTPUT
        Blkdata() //  block of message data

Begin
Set Step=8
Loop s=1 to szblk
  LL(s) = ABS (LL(s))
  LLn(s) = step* round (LL(s) / step)
  BD = LL(s) - LLn(s)
  If BD <= 0 then Blkdata(s) = 0 else Blkdata(s) = 1
End Loop
End
```

3.2.5 Secret File

In this section the blocks that as an output from the previous section. After that merge the blocks and convert each eight bits to form one byte as illustrated in algorithm (3.19), and constructed the block of bytes that saved in file.

At the beginning of the extraction stage the type of secret message is determine then the file of secret message is created, (of same type like the original):

- For text file type convert each byte to characters and put in the file. As illustrated in algorithm (3.20).
- For image file type built the header of BMP image file and then put the header in the file followed by the data. Constructed this process is illustrate in algorithm (3.21).

A detailed description of the BMP file format is presented in appendix (B).

- In audio file type built the header of Wav type and at the first put the header in the file and then the data. As illustrated in algorithm (3.22).

A detailed description of the WAV file format is presented in appendix (A).

Algorithm 3.19 convert binary ASCII code to byte value

```
INPUT
        B () // Array of one dimension represent the binary code of data
        IB  // the size of B array (in bits)
OUTPUT
        X () // Array of one dimension represent the secret data
        N   // number of characters

Begin
Set Bcount = 0
Set N = 0
Loop while (Bcount < IB)
  Set bits = 1
  Set X (N) = 0
Loop For k = 0 to 7
    X(N) = X(N) + B(k) * bits
    Set bits = bits * 2
  End Loop
  N = N + 1
  Bcount = Bcount + 8
End Loop While
End
```

Algorithm 3.20 Create Text file

```
INPUT
        X () // Array of one dimension represent the secret data
        szx  // size of array X
OUTPUT
         The Text file
Begin
Open Text file for write
Loop For i=1 to szx
  CH=convert (X (i)) to character
  Write the CH in Text file
End Loop
End
```

Algorithm 3.21 Create BMP file

```
INPUT
        X () // Array of one dimension represent the secret data
        szx  // size of array X
OUTPUT
        The BMP file
Begin
Open BMP file for write
Write BMP header
Loop For i=1 to szx
  Write X (i) in BMP file
End Loop
End
```

Algorithm 3.22 Create WAV file

```
INPUT
        X () // Array of one dimension represent the secret data
        szx  // size of array X

OUTPUT
        The WAV file

Begin
Open WAV file for Write
Write WAV header
Loop For i=1 to Szx
  Write X (i) in WAV file
End Loop
End
```

3.3 Experimental Result & System Evaluation

Most steganography systems require that the communication must be invisible, such that an expected attacker cannot know if there is an embedded message inside the stego-object. In fact, imperceptibility of the stego AVI reflects how much it is affected due to the embedding process, in other word, imperceptibility can be decided by measuring that effect. In the proposed stego-system, the MSE and PSNR measurement is adopted.

The test will be done on the two types of methods by using three different types of secret files (text, image, audio) and AVI cover files. At the beginning of the project separate the AVI file into audio and video streams by using “Ulead Media Studio Pro 6.0”. Then the test will be done after embedding on video and audio stream.

- In the first case study: the test done on the two AVI cover file and five samples of secret file. The first one is text whose size is 6.92KB, two image samples were used, the type of each one are BMP 24bits true color, the first one (PIC1) 256×256 pixels and the size is 192KB, and the second one (PIC2) 352×240 pixels and the size is 247KB. For audio two samples were used for testing (speech, music) both are WAV type PCM, mono, 8bits. The speech is 8sample rate and the size 37.8KB, and the music sample has 22050sample rate and the size is150KB.

  For AVI cover, two samples were used (AVI1, AVI2) they have same format but differ in number of frames and in audio format, in AVI1 the video stream has 75 frames and its size is 29.0 MB, the audio stream has 22050 sample rate, stereo, 16bits and the size is 258KB. In AVI2 the video stream has 117 frames and its size is 45.2MB, the audio stream has 16000sample rate, stereo, 8bits and its size is 146KB.

  The two types of hiding methods were used to hide in video, they are (Least significant bit, Haar Wavelet transformed), in first hiding method the hiding is done in the least position is performed, while in the second method used one level wavelet decompose. The method were used to hide in audio, it is least significant bit that hides limited block of data in limited bytes.

  The results are shown in two tables (3.1 and 3.2), table (3.1) for video stream and table (3.2) for audio stream.

Table (3.1) the test results for the hiding methods in video stream

Method type	Cover video file	Secret file	Size of secret file	Number of frame used	Cover-Stego_cover	
					MSE	PSNR
LSB	AVI1 29MB	Text	6.92KB	1	1.123	47.627
		PIC1	192KB	16	1.941	45.250
		PIC2	247KB	21	1.898	45.349
		Speech	37.8KB	4	1.526	46.296
		Music	150KB	13	1.873	45.405
	AVI2 45.2MB	Text	6.92KB	1	1.107	47.691
		PIC1	192KB	16	1.945	45.241
		PIC2	247KB	21	1.835	45.494
		Speech	37.8KB	4	1.522	46.308
		Music	150KB	13	1.872	45.407
HWT	AVI1 29MB	Text	6.92KB	3	2.316	44.483
		PIC1	192KB	63	3.158	43.136
		PIC2	247KB	The cover size is not enough		
		Speech	37.8KB	13	3.009	43.346
		Music	150KB	49	3.120	43.188
	AVI2 45.2MB	Text	6.92KB	3	2.212	44.683
		PIC1	192KB	63	3.217	43.056
		PIC2	247KB	81	2.867	43.557
		Speech	37.8KB	13	3.062	43.272
		Music	150KB	49	3.184	43.102

Table (3.2) the test results for the hiding method in audio stream

Method type	Cover audio file	Overhead data	Number of frame used	Cover-Stego_cover	
				MSE	PSNR
LSB	AVI1 256KB	Text	1	0.000034	92.814
		PIC1	16	0.000049	91.217
		PIC2	21	0.000042	91.943
		Speech	4	0.000038	92.357
		Music	13	0.000053	90.896

	AVI2 146KB	Text	1	0.000013	96.875
		PIC1	16	0.000008	89.093
		PIC2	21	0.000073	89.471
		Speech	4	0.000087	88.746
		Music	13	0.000094	88.424
HWT	AVI1 258KB	Text	3	0.000008	99.400
		PIC1	63	0.000023	94.575
		PIC2	The cover size is not enough		
		Speech	13	0.000011	97.585
		Music	49	0.000019	95.367
	AVI2 146KB	Text	3	0.000013	96.875
		PIC1	63	0.000004	92.104
		PIC2	81	0.000002	95.114
		Speech	13	0.000002	95.114
		Music	49	0.000034	92.896

- In the second case study: same samples of AVI files were used in addition to the five secret files, but the difference is in distributing the secret data file on fit plus ten frames. And show the different results.

Table (3.3) the test results for the hiding methods in video by adding 10 frames

Method type	Cover video file	Secret file	Size of secret file	Number of frame used	Cover-Stego_cover	
					MSE	PSNR
LSB	AVI1 29MB	Text	6.92KB	11	0.101	58.109
		PIC1	192KB	26	1.196	47.353
		PIC2	247KB	31	1.282	47.052
		Speech	37.8KB	14	0.437	51.722
		Music	150KB	23	1.059	47.883
	AVI2 45.2MB	Text	6.92KB	11	0.102	58.045
		PIC1	192KB	26	1.205	47.321
		PIC2	247KB	31	1.264	47.112
		Speech	37.8KB	14	0.439	51.706
		Music	150KB	23	1.060	47.876

HWT	AVI1 29MB	Text	6.92KB	13	0.541	50.801
		PIC1	192KB	73	2.728	43.772
		PIC2	247KB	The cover size is not enough		
		Speech	37.8KB	23	1.668	45.856
		Music	150KB	59	2.592	43.995
	AVI2 45.2MB	Text	6.92KB	13	0.537	50.832
		PIC1	192KB	73	2.789	43.676
		PIC2	247KB	91	2.551	44.064
		Speech	37.8KB	23	1.722	45.771
		Music	150KB	59	2.636	43.922

Table (3.4) the test results for the hiding methods in audio

Method type	Cover audio file	Overhead data	Number of frame used	Cover-Stego_cover	
				MSE	PSNR
LSB	AVI1 256KB	Text	1	0.000034	92.814
		PIC1	16	0.000049	91.217
		PIC2	21	0.000042	91.943
		Speech	4	0.000038	92.357
		Music	13	0.000053	90.896
	AVI2 146KB	Text	1	0.000013	96.875
		PIC1	16	0.000008	89.093
		PIC2	21	0.000073	89.471
		Speech	4	0.000087	88.746
		Music	13	0.000094	88.424
H WT	AVI1 258KB	Text	3	0.000008	99.400
		PIC1	63	0.000023	94.575
		PIC2	The cover size is not enough		
		Speech	13	0.000011	97.585
		Music	49	0.000019	95.367
	AVI2 146KB	Text	3	0.000013	96.875
		PIC1	63	0.000004	92.104
		PIC2	81	0.000002	95.114
		Speech	13	0.000002	95.114
		Music	49	0.000034	92.896

3.4 Results Discussion

From the table of results (3.1, 3.2, 3.3, 3.4) the following points are notice:

1. In the first step, the AVI file separated into video and audio parts. This video part consider as streams of images, so the method of hiding information used in this project is to hide the given information into these image frames. So the selected method of hiding as it appears in the tables are the Least Significant Bit and Haar Wavelet Transform.

   Three types of information (text, image, audio) are used to hide into the video stream. The computation started by calculating the minimum number of frames that enough to hide the given information. This number of frames depends on the method of hiding and the size of information to be hided. Then to increase the security and robustness of the method, the number of frames is increased if there are enough extra unused frames.

   From the tables it appears for both methods of hiding the MSE reduced and the PSNR increased when the number of frame used are increased, but even with the minimum number of frames the MSE are reasonably small and PSNR are reasonably large, because the cover size are large in comparison to the hided information.
2. On the other audio part small size of data will be hidden, this hided data in this part is so important. So to increase the security either by chosen for example a random location or encrypted the hidden data.

CHAPTER TWO

Steganography Concept

2.1 Introduction

Information hiding in digital images, video or audio had drawn much attention in recent years. Some auxiliary information is implicitly combined with a piece of multimedia data, i. e. the host signal, to form a composite signal for certain interesting applications [Jay03].

Data hiding techniques should be capable of embedding data in a host signal with the following restrictions and features [Pol01]:

- The host signal should be non-objectionally degraded and the embedded data should be minimally perceptible. That means is the observer should not be able to notice the presence of the data even if it were perceptible.
- The embedded data should be directly encoded into the media rather than into a header or a wrapper so that the data remain intact across varying data file formats.
- The embedded data should be immune to modifications ranging from intentional and intelligent attempts at removal to anticipated manipulations. e.g. channel noise, re-sampling, cropping, etc…
- Asymmetrical coding of the embedded data is desirable since the purpose of data hiding is to keep the data in the host signal but not necessarily to make the data difficult to access.
- The embedded data should be self clocking or arbitrarily re-entrant. This ensures that the embedded data can be recovered even when only fragments of information are available.

Both ***Steganography*** and ***Watermarking*** describe techniques that are used to imperceptibility convey information by embedding it into the cover-data [Kaz01]. Digital steganography has become increasingly used in recent years. Steganography literally means, “Covered writing” and includes the methods of transmitting secret messages through cover carriers in such a manner that the existence of the embedded messages is undetectable [Civ01].

2.2 Steganography

Steganography is the art of hiding and transmitting data through apparently innocuous carriers in an effort to conceal the existence of the data. Though steganography is an ancient craft, the onset of computer technology has given it a new life. Computer-based steganographic techniques introduce changes to digital covers to embed information foreign to the native covers. Such information may be communicated in the form of text, binary files, or provide additional information about the cover and its owner such as digital watermarks or fingerprints [Wat01].

It differs from digital watermarking for its information different requirements on *imperceptibility* (including both visual and statistical imperceptibility), *robustness* (robust imperceptibility against cover modifications), *security* (how easy it is to break the message), and *capacity* (how much information can be embedded hidden in a certain media). Steganography needs to achieve large capacity, high security level, and high imperceptibility, but does not have to be robust against cover modifications. While digital watermarking has relatively low requirements on security and capacity, it must be robust against imperceptibility, both faults (e. g. noise) and malicious faults (e. g. attacks like accidental scaling of the cover)[Sny02].

2.2.1 Steganography Model

Most applications of steganography follow one general principle, illustrated in Fig. 2.1. Alice (in the field of cryptography, communication protocols usually involve two fictional characters named Alice and Bob or use a name whose first character matches the first letter of their role (e.g. Wendy the warden)), who wants to share a secret message M with Bob, randomly chooses (using private random source r) a harmless message C, called cover-object, which can be transmitted to Bob without raising suspicion, and embeds the secret message into C, probably by using a key K, called stego-key. Alice therefore changes the cover C to a stego-object S. This must be done in a very careful way, so that a third party, knowing only the apparently harmless message S, cannot detect the existence of the secret [Kat00].

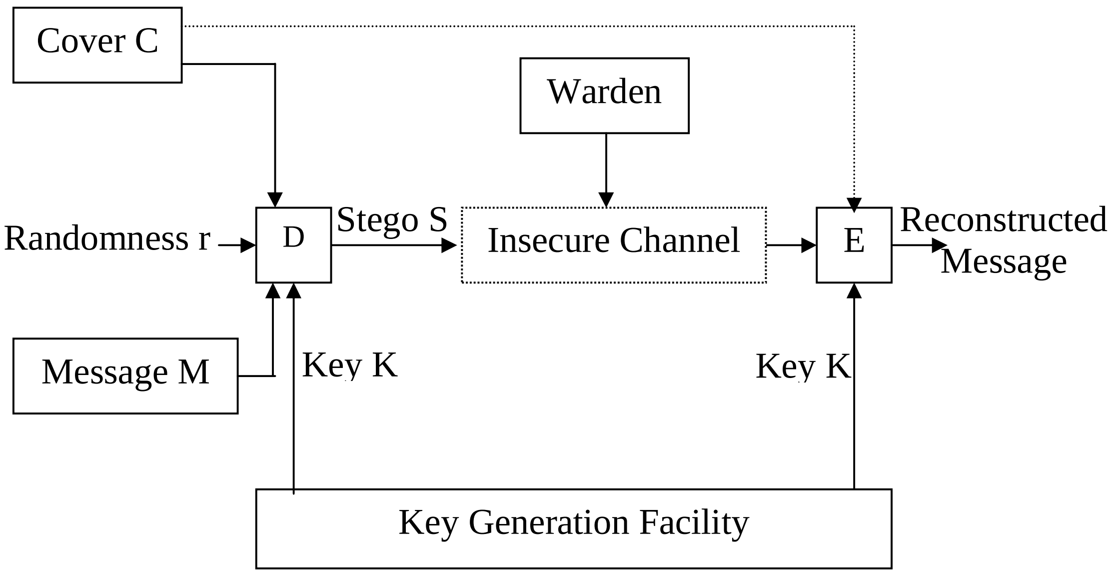


Fig. 2.1 Schematic description of steganography

Cover-object: refers to the object used as the carrier to embed messages into. Many different objects have been employed to embed messages into for example images, audio, and video as well as file structures, and html pages to name a few. Stego-object: refers to the object, which is carrying a hidden message. So given a cover object, and a messages the goal of the steganographer is to produce a stego object which would carry the message.

Alice then transmits S over an insecure channel to Bob and hopes that Wendy will not notice the embedded message. Bob can reconstruct M since the embedding method used by Alice is known and Bob has access to the key K used in the embedding process [Kha04].

A third party watching the communication should not be able to decide whether the sender is active in the sense that the sender sends covers containing secret message rather than covers without additional information. Formally, if an observer has access to the cover-objects transmitted between both communication parties, the observer should not be able to decide which cover-object contains secret information. Thus, the security of invisible communication lies mainly in the inability to distinguish cover-object from stego-object. Obviously, a cover should not be used twice, since an attacker who has access to two “versions” of one cover can easily detect and possibly reconstruct the message. To avoid accidental reuse, both sender and receiver should destroy all covers they have already used for information transfer [Kat00].

2.2.2 Steganography vs. Cryptography

Steganography is not the same as cryptography. In cryptography, the structure of a message is changed to render it meaningless and unintelligible unless the decryption key is available. Cryptography makes no attempt to disguise or hide the encoded message. Steganography does not alter the structure of the secret message, but hides it inside a cover. It is possible to combine the two techniques by encrypting a message using cryptography and then hiding the encrypted message using steganography. The resulting stego-image can be transmitted without revealing that secret information is being exchanged. Furthermore, even if an attacker were to defeat the steganographic technique and detect the message from the stego-image, there would still the need for the cryptographic decoding key to decipher the encrypted message [Lin03].

2.2.3 Application of Steganography [Lin03]

There are many applications for digital steganography, including copyright protection, feature tagging, and secret communications.

Copyright Protection: A secret copyright notice or watermark can be embedded inside an image to identify it as intellectual property. This is the watermarking scenario where the message is the watermark. The "watermark" can be a relatively complicated structure. In addition, when an image is sold or distributed an identification of the recipient and time stamp can be embedded to identify potential pirates. A watermark can also serve to detect whether the image has been subsequently modified. Detection of an embedded watermark is performed by a statistical, correlation, or similarity test, or by measuring other quantity characteristic to the watermark in a stego- image. The insertion and analysis of watermarks to protect copyrighted material is responsible for the recent surge of interest in digital steganography and data embedding.

Feature Tagging: Captions, annotations, time stamps, and other descriptive elements can be embedded inside an image, such as the names of individuals in a photo or locations in a map. Copying the stego- image also copies all of the embedded features and only parties who possess the decoding stego- key will be able to extract and view the features. In an image database, keywords can be embedded to facilitate search engines. If the image is a frame of a video sequence, timing markers can be embedded in the image for synchronization with audio. The number of times an image has been viewed can be embedded for “pay- per-view” applications.

Secret Communications: In many situations, transmitting a cryptographic message draws unwanted attention. The use of cryptographic technology may be restricted or forbidden by law. However, the use steganography does not advertise covert communication and therefore avoids scrutiny of the sender, message, and recipient. A trade secret, blueprint, or other sensitive information can be transmitted without alerting potential attackers or eavesdroppers.

2.3 Steganography in AVI file

In modern time, digital images, audio files, and streaming video have become carries for hidden information, while our networks are high-speed delivery channels [Hos03].

AVI stands for Audio Video Interleave. It a special case of the RIFF (Resource Interchange File Format). Microsoft defines AVI. AVI is the most common format for audio video data used in computer [Mcg97].

2.3.1 Video steganography

The video stream in an AVI file is nothing more than a sequence of bitmaps. Video steganography is not too different from image steganography. It can be said that video steganography is a derivative of image steganography; this is because video is made up of a series of images that are transmitted. So whatever techniques (and attacks) can be applied to images also apply for videos [Pot03].

The embedded information must be perceptually invisible to ensure high visual quality. By embedding the information directly into the pixels of each frame, rather than as a header of the file, the information will not be so easily lost when the video format is changed or the video is cropped, etc. Furthermore, if independent files are used, they may require synchronization during playback, may easily be separated, or additional storage space could be required.

Since a video sequence can be broken down into a series of still images, we will begin our discussion by presenting a robust technique for image steganography [Lan00].

2.3.2 Image steganography

Information can be hidden many different ways in images. To hide information, straight message may encode every bit of information in the image insertion or selectively embed the message in “noisy” areas that draw less attention— those areas where there is a great deal of natural color variation. The message may also be scattered randomly throughout the image. Redundant pattern encoding “wallpapers” the cover image with the message. A number of ways exist to hide information in images. Common approaches include digital

- Least significant bit insertion
- Masking and filtering
- Transformation technique

Each of these techniques can be applied, with varying degrees of success, to different image files [Joh98][Sel00].

A. Least significant bit insertion [Joh98][Sel00]

Least significant bit (LSB) insertion is a common, simple approach to embedding information in a cover file. Unfortunately, it is vulnerable to even a slight image manipulation. Converting an image from a format like GIF or BMP, which reconstructs the original message exactly *(lossless compressio*n) to a JPEG, which does not *(lossy compressio*n), and then back could destroy the information hidden in the LSBs.

24- bit images to hide an image in the LSBs of each byte of a 24- bit image, you can store 3 bits in each pixel. A 1024 x 768 image has the potential to hide a total of 2,359,296 bits (294,912 bytes) If you compress the message to be hidden before you embed it, you can hide a large amount of information. To the human eye, the resulting stego- image information will look identical to the cover image. For example, the letter A can be hidden in three (assuming no compression). For example if the original raster pixels data for 3 pixels (9 bytes) are:

(00100111 11101001 11001000)

(00100111 11001000 11101001)

(11001000 00100111 11101001)

The binary value for A is **10000011**. Inserting the binary value for A in the three pixels would result in

(00100111 11101000 11001000)

(00100110 11001000 11101000)

(11001000 00100111 11101001)

The underlined bits are the only three actually changed in the 8 bytes used. On average, LSB requires that only half the bits in an image be changed. You can hide data in the least and second least significant bits and still the human eye would not be able to discern it.

8- bit images are not as forgiving to LSB manipulation because of color limitations. Steganography software authors have devised several approaches—some are more successful than others— to hide information in 8- bit images. First, the cover image must be more selected so that the stego-image will not broadcast carefully the existence of an embedded message.

When information is inserted into the LSBs of the raster data, the pointers to the color entries in the palette are changed. In an abbreviated example, a simple four- color palette of white, red, blue, and green has corresponding palette position entries of 0 (00), 1 (01), 2 (10), and 3 (11), respectively. The raster values of four adjacent pixels of white, white, blue, and blue are 00 00 10 10. Hiding the binary value 1010 for the number 10 changes the raster data to 01 00 11 10, which is red, white, green, blue. These gross changes in the image are visible and clearly highlight the weakness of using 8- bit images. On the other hand, there is little visible difference noticed between adjacent gray values.

B. Masking and filtering [Joh98]

Masking and filtering techniques, usually restricted to 24- bit and gray-scale images, hide information by marking an image, in a manner similar to paper watermarks. Traditional steganography *conceals* information; Watermarks *extend* information and become an attribute of the cover image. Digital watermarks may include such information as copyright, ownership, or license. In steganography, the object of communication is the hidden message. In digital watermarks, the object of communication is the cover.

Masking is more robust than LSB insertion with respect to compression, cropping, and some image processing. Masking techniques embed information in significant areas so that the hidden message is more integral to the cover image than just hiding it in the "noise " level. This makes it more suitable than LSB with, for instance, lossy JPEG images.

C. Transformation technique

Another class of techniques is embedding the message by modulating coefficients in a transform domain, such as the Discrete- Cosine Transform (DCT) (used in JPEG compression), Discrete Fourier Transform, or Wavelet Transform. Transform techniques can offer superior robustness against lossy compression because they are designed to resist or exploit the methods of popular lossy compression algorithms. An example of a transform- based steganographic system is the "Jpeg- Jsteg" software, which embeds the message by modulating DCT coefficients of the stego- image based upon bits of the message and the round- off error during quantization. Transform-based steganography also typically offer increased robustness to scaling and rotations or cropping, depending on the invariant properties of a particular transform [Lin03].

2.3.3 Audio steganography

Because of the range of the human auditory system (HAS), data hiding in audio signals is especially challenging. The HAS perceives over a range of power greater than one billion to one and range of frequencies greater than one thousand to one. Also, the auditory system is very sensitive to additive random noise. Any disturbances in a sound file can be detected as low as one part in ten million (80 dB below ambient level). However, while the HAS has a large dynamic range, it has a fairly small differential range-large sounds tend to drown quiet sounds. When performing data hiding on audio, one must exploit the weaknesses of the HAS, while at the same time being aware of the extreme sensitivity of the human auditory system. The method of the encoded message in audio is:

A. Low bit encoding: is the simplest way to embed data into other data structures. By replacing the least significant bit (LSB) of each sampling point by a coded binary string (see Fig. 2.2), we can encode a large amount of data in an audio signal. Ideally the channel capacity is 1 kb per second per kHz; so for example, the channel capacity would be 44 kbps in a 44 kHz sampled sequence. Unfortunately, this introduces audible noise.

The major disadvantage of this method is poor immunity to manipulation. Encoded information can be destroyed by channel noise, resembling, etc., unless it is encoded using redundancy techniques. In order to be robust, these techniques reduce the data rate which could result in the requirement of a host of higher magnitude, often by one to two orders of magnitude. In practice, this method is useful only in physical storage and closed digital-to-digital environment [Pol01].

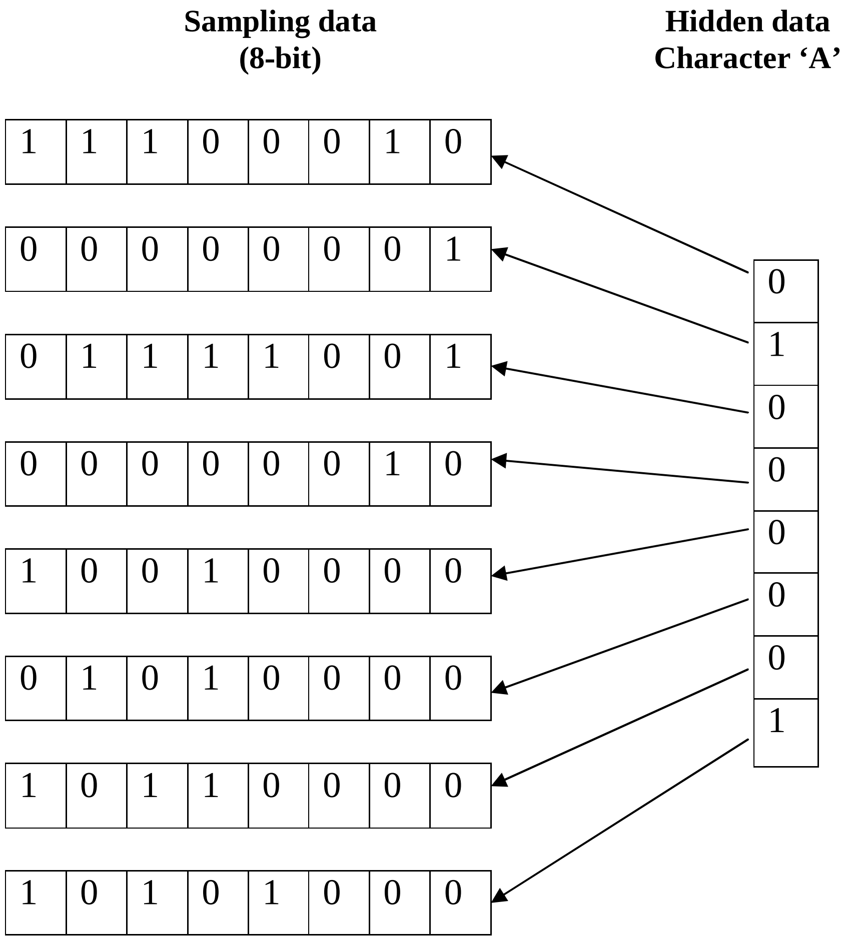


Fig. 2.2 Encoding character 'A' in a stream of audio samples

B. Phase coding: The phase coding method works by substituting the phase of an initial audio segment with a reference phase that represents the data. The phase of subsequent segments is adjusted in order to preserve the relative phase between segments. The phase coding, is one of the most effective coding methods in terms of the signal to perceived noise ratio. When the phase relation between each frequency components is dramatically changed, noticeable phase dispersion will be occur. However, as long as the modification of the phase is sufficiently small an inaudible coding can be achieved. The phase coding method works by substituting the phase of an initial audio segment with a reference phase that represents the data[Rab04][Sel00].

C. Spread spectrum: Most communication channels try to concentrate audio data in as narrow a region of the frequency spectrum as possible in order to conserve bandwidth and power. When using a spread spectrum technique, however, the encoded data is spread across as much of the frequency spectrum as possible. One particular method is the Direct Sequence Spread Spectrum (DSSS) encoding, spreads the signal by multiplying it by a certain maximal length pseudorandom sequence, known as a chip. The sampling rate of the host signal is used as the chip rate for coding. The calculation of the start and end quanta for phase locking purposes is taken care of by the discrete sampled nature of the host signal. As a result, a higher chip rate and, therefore, a higher associated data rate are possible. However, unlike phase coding, DSSS does introduce additive random noise to the sound [Rab04].

D. Echo data hiding: Echo data hiding embeds data into a host signal by introducing an echo. The data are hidden by varying three parameters of the echo, these are: initial amplitude, decay rate and offset, or delay characteristics. As the offset between the original and the echo decreases, the two signals blend. At a certain point, the human ear cannot distinguish between the two signals and the echo is merely heard as added resonance. This point depends on factors such as the quality of the original recording, the type of sound and the listener [Pol01][Sel00].

2.4 Wavelet Transforms [Han98]

Wavelet transforms are based on a relatively new concept. There is a push toward the use of wavelets in signal processing and analysis in place of (or in addition to) the Discrete Cosine Transform (DCT), which is used in the JPEG standard for image compression. Recently, many algorithms have been proposed to use wavelets for image compression. The techniques that are currently being used in working with images can be generalized for use with wavelet transforms. The involve d wavelet algorithms is the simple wavelet transform called the Haar transform

2.4.1 Discrete Wavelet Transform (DWT)

The DWT has a *scaling function* and a *wavelet function* associated with it. The scaling function can be implemented using a low pass filter and is to create the scaling coefficients that represent the signal approximation. The wavelet function can be implemented as a high pass filter and is used to create the wavelet coefficients that represent the signal details. If the DWT is used by scaling and shifting by powers of two, the signal will be well represented and the decomposition will be efficient and easy to compute. In order to apply the DWT to images, combinational of the filters (combinations of the scaling function and the wavelet function) are used first along the rows and then along the columns to produce unique subbands [Jac03].

The LL subband is produced by low pass filtering along the rows and columns and is commonly referred to as a course approximation of the image because the edges tend to smooth out. The LH subband is produced by low pass filtering along the rows and high pass filtering along columns, thus capturing the horizontal edges. The HL subband is produced by high pass filtering along the rows and low pass filtering along columns, thus capturing the vertical edges. The HH subband is produced by high pass filtering along the rows and columns, thus capturing the diagonal edges. The LH, HL and HH subbands together are called the detail subbands. These subbands are shown in Fig 2.3. By repeating the process on the LL subband, additional scales are produced. In this context scales are synonymous to the detail subbands [Jac03].

The equations for discrete wavelet decomposition and discrete wavelet reconstruction will show in the next two sections.

LL	**HL**
LH	**HH**

Fig 2.3 The four subbands wavelet transform image.

- **Discrete Wavelet Decomposition [Bur98]**

A signal x of length N can be decomposed in any level to give a coarser approximation of the signal in the next level. The approximation coefficients at level j is given by:

$$c_j(k) = \sum_m h(m-2k)c_{j+1}(m) \qquad (2.1)$$

$$d_j(k) = \sum_m g(m-2k)c_{j+1}(m) \qquad (2.2)$$

Where c are called the scaling function or the approximation coefficients and d are called the wavelet or the details coefficients. h, g are both finite even length discrete values wavelet filters are called the decomposition low-pass and high-pass wavelet filters respectively.

Assuming that c with the highest resolution subscript is the original input signal. At each stage of the decompositions (2.1) and (2.2), the length of the resulting signals c_j and d_j is half the length of c_{j+1} because of the down-sampling process after each time in which the decomposition occurs.

The down-sampler (sometimes called a sampler or decimator) takes a signal x(n) as an input and produces an output y(n)=x(2n). The down-sampler is symbolically shown in Fig. 2.4.

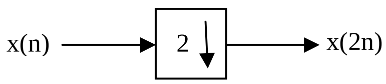


Fig 2.4 The Down-Sampler

In down-sampling, there is clearly the possibility of losing information since half of the data is discarded. The scale-j coefficients are filtered with the two low-pass h and high-pass g filters, after which down-sampling gives the next coarser scaling and wavelet coefficients. Both of them, having lengths equal to half the length of the input signal. In (Fig. 2.5), the decomposition process is depicted for two decomposition stages.

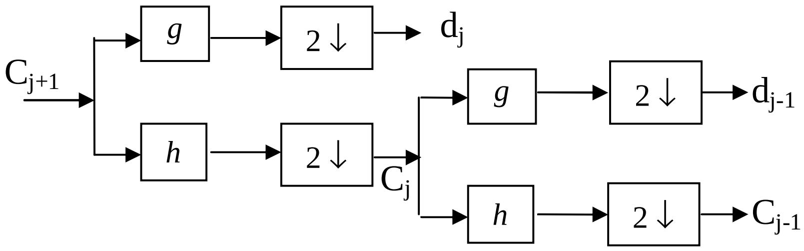


Fig 2.5 Two-Stage decomposition process

- **Discrete Wavelet Reconstruction [Bur98]**

A signal considered at a resolution j+1, can be reconstructed from the combination of the scaling function and wavelet coefficients at a coarser resolution j. This can be written as:

$$c_{j+1}(k) = \sum_m c_j(m)\bar{h}(k-2m) + \sum_m d_j(m)\bar{g}(k-2m) \qquad (2.3)$$

Where $\bar{h}, \bar{g}$ are both finite even length discrete values wavelet filters called the reconstruction low-pass and high-pass wavelet filters respectively and are derived directly from the low-pass and high-pass decomposition filters.

At each stage of the reconstruction process (2.3), the length of the resulting signals c_{j+1} equals to the sum of the length of both c_j and d_j because of the up-sampling process after each time in which the reconstruction occurs.

The up-sampling means that the input to the filter has zeros inserted between each of the original terms. In other words y(2n)=x(n) and y(2n+1)=0.

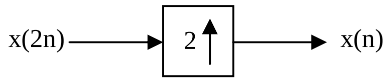


Fig 2.6 The Up-Sampler

The input signal is stretched to twice its original length and zeros are inserted. The up-sampler is symbolically shown in Fig. 2.6.

In (Fig. 2.7), the reconstruction process is depicted for two reconstruction stages.

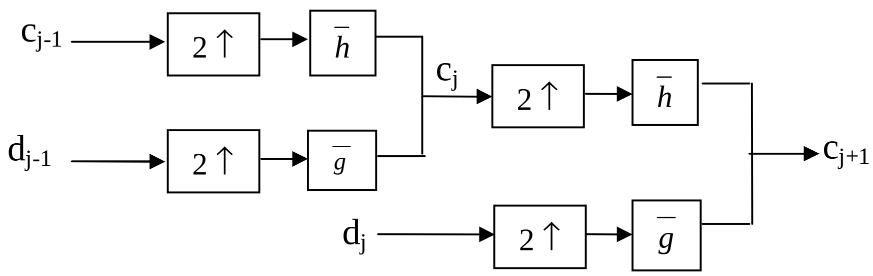


Fig 2.7 Two-Stage reconstruction process

2.4.2 Haar Wavelet Transform [Sal00]

The Haar transform is one of the simplest transforms in wavelet mathematics. The Decomposition and Reconstruction phases are described by:

- **Decomposition (DHWT):** Given an image I: [1, M_r] × [1, M_c] where M_r, M_c is the numbers of rows and columns. The DHWT and RHWT for a one-dimensional signal can be also described in the form of two-dimensional signal. The DHWT and RHWT for two dimensional images can be similarly defined by implementing the one dimensional DHWT and RHWT for each dimension M_r and M_c separately: DHWT M_c [DHWT M_r [I (M_r, M_c)]], Consider the value of two neighboring pixels in each row (and the same on the column) I(2i-1) and I(2i). The

DHWT maps the original image I onto a low pass image L and high pass image H.

$$L(i) = \frac{I(2i-1) + I(2i)}{\sqrt{2}} \quad \text{For i=1 to vector length/2} \quad (2.4)$$
$$H(i) = \frac{I(2i-1) - I(2i)}{\sqrt{2}}$$

- **Reconstruction (RHWT):** The reconstructed HWT equation are

$$I(2i-1) = \frac{L(i) + H(i)}{\sqrt{2}} \quad \text{For i=1 to vector length/2} \quad (2.5)$$
$$I(2i) = \frac{L(i) - H(i)}{\sqrt{2}}$$

2.5 Fidelity Measures [Toz03]

The well known image quality measure; Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR) have been used in order to objectively evaluate the performance of the proposed method.

The MSE is found by taking the summation of the square of the difference original (image, audio) and the stego(image or audio) and finally dividing it by the total number of samples as shown below:

$$MSE = \frac{1}{size} \sum_{i=1}^{size} \left(O_i - S_i \right)^2 \quad (2.7)$$

Where size is the total number of the samples in the horizontal and vertical dimension of the image. O_i represent the original (image or audio) and S_i represent the stego(image or audio).

The quality image of PSNR is defined with:

$$PSNR = 10 \log_{10} \left(\frac{I_{\max}^2}{MSN} \right) \tag{2.8}$$

Where I_{max} is equal to 255 for 8 bit.

2.6 AVI File

The Microsoft Audio/Video Interleaved (AVI) file format is a Resource Interchange File Format (RIFF) file specification used with applications that capture, edit, and playback audio/video sequences. In general, AVI files contain multiple streams of different types of data. Most AVI sequences will use both audio and video streams. A simple variation for an AVI sequence uses video data and does not require an audio stream. Specialized AVI sequences might include a control track or Musical Instrument Digital Interface (MIDI) track as an additional data stream. The control track could control external devices such as an Media Control Interface (MCI) videodisc player. The MIDI track could play background music for the sequence. While a specialized sequence requires a specialized control program to take advantage of all its capabilities, applications that can read and play AVI sequences can still read and play an AVI sequence in a specialized file. (These applications ignore the non-AVI data in the specialized file.) [Avi99]

2.6.1 RIFF Files [Mcg97]

RIFF files are built from

(1) RIFF Form Header

'RIFF' (4 byte file size) 'xxxx' (data)

where 'xxxx' identifies the specialization (or form) of RIFF. 'AVI ' for AVI files.

where the data is the rest of the file. The data is comprised of chunks and lists. Chunks and lists are defined immediately below.

(2) A Chunk

(4 byte identifier) (4 byte chunk size) (data)

The 4 byte identifier is a human readable sequence of four characters such as 'JUNK' or 'idx1'

(3) A List

'LIST' (4 byte list size) (4 byte list identifier) (data)
where the 4 byte identifier is a human readable sequence of four characters such as 'rec ' or 'movi'
where the data is comprised of LISTS or CHUNKS.

2.6.2 AVI RIFF Form

AVI files use the AVI RIFF form. The AVI RIFF form is identified by the four-character code "AVI ". All AVI files include two mandatory LIST chunks. These chunks define the format of the streams and stream data. AVI files might also include an index chunk. This optional chunk specifies the location of data chunks within the file. An AVI file with these components has the following form:

```
RIFF ('AVI '
    LIST ('hdrl'
          .
          .
          .
        )
    LIST ('movi'
          .
          .
          .
        )
    ['idx1'<AVI Index>]
    )
```

The LIST chunks and the index chunk are subchunks of the RIFF "AVI " chunk. The "AVI " chunk identifies the file as an AVI RIFF file. The LIST "hdrl" chunk defines the format of the data and is the first required list chunk. The LIST "movi" chunk contains the data for the AVI sequence and is the second required list chunk. The "idx1" chunk is the optional index chunk. AVI files must keep these three components in the proper sequence. The LIST "hdrl" and LIST "movi" chunks use subchunks for their data [Avi99]. A detailed description of AVI file structure is present in appendix (C).

Table of Contents

Information

Name: Ghessaq Hussein Ali Al-Anbaki

Al-Nahrain University, College of Science, Computer Science Dept.

Discussion Date: 8/9/2005

M.Sc. Thesis

Thesis Title: Steganography in AVI Files

Supervisors: Dr. Taha Saadon Bashaga

Address: District 883 / Lane 49 /House No. 20
Al-Jihad Square – Baghdad – Iraq.

Phone No.: Ground : 5546361
Mobile : 07702659571

List of Abbreviations

AVI	Audio Video Interleave
BMP	Bit-Map image file
dB	decibell
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DSSS	Direct sequence spread spectrum
DHWT	Decomposition Haar Wavelet Transform
DWT	Discrete Wavelet Transform
GIF	Graphics Interchange Format
HAS	Human Auditory System
hdrl	header list
HH	high high pass filtring
HL	high low pass filtring
idxl	index list
JPEG	Joint Photographic Experts Group
JPEG-2000	Joint Photographic Experts Group 2000
LH	low high pass filtring
LL	low low pass filtring
LSB	Least Significant Bit
LS2B	Least Significant second Bit
MCI	Media Control Interface
MIDI	Musical Instrument Digital Interface
mod	Modulus
MPEG	Motion Pictures Experts Group
MSE	Mean Square Error
PCM	Pulse Code Modulation
PSNR	Peak Signal-to-Noise Ratio
RGB	Red Green Blue
RHWT	Reconstruction Haar Wavelet Transform
RIFF	Resource Interchange File Format
WAV	Windows Audio Visual
WT	Wavelet Transform

List of Symbols

c	Approximation coefficient
C	Cover-object
d	Details coefficient
g	High-pass filter
h	Low-pass filter
H	High pass image
$\mathbf{I(M_r, M_c)}$	Two dimensional images
K	Stego-key
L	Low pass image
m_i	i^{th} message bit
$\mathbf{M_c}$	Numbers of columns
$\mathbf{M_r}$	Numbers of rows
x[n], y[n]	Discrete signals

References

[Bur98] C. S. Burrus, **R. A. Gopinath**, **and H. Guo**, " Introduction to Wavelets and Wavelet Transforms: A Primer", Prentice Hall, Inc., 1998.

[Cac98] C. Cachin, "An Information – Theoretical Model for Steganography", MIT Laboratory for computer science, 545 Technology Square, Cambridge, MA 02139, USA, May-13-1998.

[Cha00] J. J. Chae and B. S. Manjunath, "Data Hiding in Video", Department of Electrical and Computer Engineering, University of California, Santa Barbara, 2000.

[Civ01] P. Civicioglu and M. Alci, "Hiding Information in Images", Erciyes University, 38039, Kayseri, Turkey, 2001.

[Dor03] Dorian A. Flowers," Steganography ", Xavier University of Louisiana, Computer Science and Computer Engineering, 1 Drexel Drive, Campus Box 50, 504.520.5248, 2003.

[Han98] Han-Yang Lo, S. Topiwala, J. Wang," Wavelet Based Steganography and Watermarking", Cornell University, Computer Science Department, Multimedia: CS 631,spring 1998.

[Hos03] C. Hosmer and C. Hyde, "Common Sense Security for Common Businesses", The Security Journal, Volume 4 – summer 2003 Edition.

[Jac03] J. T. Jackson, G. H. Gunsch, R. L. Claypoole and G. B. Lampnt, "Blind Steganography Detection Using a Computational Immune System: A Work in Progress", International Journal of Digital Evidence, Winter 2003, Issue 1, Volume4.

[Jaw05] M. J. Jawad, "Hiding Audio in Audio Using Wavelet Transform", M.Sc. Thesis, Computer Science Dept., College of Science, Al-Nahrain University, Baghdad, Iraq, 2005.

[Jay03] C. C. Jay Kuo and Po- Chyi Su, "Steganography in JPEG2000 Compressed Images"**,** IEEE Transactions on Consumer Electronics, Vol. 49, No. 4, NOVEMBER 2003

[Joh98] N.F. Johnson and S. Jajodia, "Exploring Steganography:seeing the unseen", Computing Practice, George Mason University, February 1998.

[Joh01] N. F. Johnson, **Z. Duric**, **and S. Jajodia**, "Information Hiding: Steganography and Watermarking-Attacks and Counter-measures", Kluwer Academic Publishers, 2001.

[Joh03] C. John, "Steganography IV-Reading and Writing AVI files", 2003.
http://www.rahul.net/The Code Project - Steganography IV - Reading and Writing AVI files - C# Programming.htm

[Kat00] S. Katzenbeisser, and F. A. Petitcolas, “Information Hiding Techniques for Steganography and Digital Watermarking”, Artech House, London, 2000.

[Kaz01] G. Kazakeviciute, R. Rosenbaum, “Information Hiding on Wavelet Based Schemes Under Considration of JPEG2000”, University of Rostock, Department of Computer Science, Institute of Computer Graphics Rostock, June 2001.

[Kha04] M. Kharrazi, H. T. Sencar and Nasir Memon, “Image Steganography: Concepts and Practice”, Polytechnic University, Brooklyn, NY 11201, USA, April 22, 2004.

[Lan00] D. E. Lane, “Video- in- Video Data Hiding”, Department of Electrical and Computer Engineering University of California Santa Barbara, CA 93106- 9560, 2000.

[Lin03] E. T. Lin, and E. J. Delp, “A Review of Data Hiding in Digital Images”, Video and Images Processing Laboratory (VIPER), School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, 2003.

[Mcg97] J. F. McGowan, ”AVI Overview”, 1997.
http://www.rahul.net/jfm

[Mer03] S. N. Merchant, A. Harchandani, S. Dua, H. Donde, I. Sunesara,“Watermarking of Video Data Using Integer- to- Integer Discrete Wavelet Transform”, Dept. of Electrical Engineering and of I. T. Engineering, 2003.

[Moh04] H. J. Mohammed, "Wavelet Based Information Hiding and Compression Techniques", M.Sc. Thesis, Computer Science Dept., College of Science, Al-Nahrain University, Baghdad, Iraq, 2004.

[Pet99] F. A. P. Petitcolas, R. J. Anderson and M. G. Kuhn, "Information Hiding –A Survey", proceeding of IEEE special issue on protection of multimedia control, 87(7): 1067-1078, July 1999.
http://www.cl.cam.ac.uk/~fapp2/publications/ieee99-infohiding.pdf

[Pol01] A. D. Polpitiya and W. J. Khan, "Information Hiding in Audio Files with Encryption", Washington University, USA, Version 1.0, November 2001.
http://www.netra.wustl.edu/~adpol/courses/cs502/project/REPORT.pdf

[Pot03] V. Potdar and E. Chang, "Visibly Invisible: Ciphertext as a Steganographic Carrier", School of Information System, Curtin University of Technology Perth, Western Australia, 6845, 2003
http://www.ceebi.research.cbs.curtin.edu.au

[Rab04] K. Rabah, "Steganography- The Art of Hiding Data", Department of Physics, Eastern Mediterranean University, Information Technology Journal 3 (3): 245- 269, 2004.

[Sal00] D. Salomon, "Data Compression", Department of computer science, California State University, Northridge, New York 2000.

[Sed01] F. A. Seddeq, "Image Steganography by Using Least Significant Bit Insertion Method", M.Sc. Thesis, Computer Science Dept., College of Science, Al-Nahrain University, Baghdad, Iraq, 2001.

[Sel00] D. Sellars, "An Introduction to Steganography", An internet Survey,2000. http://www.cs.uct.ac.za/courses/CS400W/NIS/paper99/dsellars/stego.pdf

[Sny02] W. E. Snyder, W. A. Sander and H. Qi, "Blind Consistency Based Steganography for Information Hiding in Digital Media", AMSRL- RO- T U. S. Army Research Office, PO Box 12211, Research Triangle Park, NC 27709- 2211, ECE Department The University of Tennessee Knoxville, TN 37996, 2002.

[Toz03] F. G. Toz, H. M. Palancioglu, E. Besdok, "Hidden Communication in Frequency Domain for Information Exchange"**,** ITU, Civil Engineering Faculty, 80626 Maslak Istanbul, Turkey, Erciyes University, Engineering Faculty, Geodesy and Photogrammetry Dept., Kayseri, Turkey, 2003.

[Wat01] J. Watkins, "Steganography - Messages Hidden in Bits", Multimedia Systems Coursework, Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK, December-15-2001.

[Wes98] A.Westfelf and G. Wolf, "Steganography in a Video Conferencing System", Dresden University of Technology, Germany. 1998.

[Avi99] "AVI Files", an Internet Survey, 1999.
http://www.user.tu_chemnitz.de/ ~noe/ video_zeng/
avi_docu/avi.htm

Republic of Iraq
Ministry of Higher Education
AL-Nahrain University
College of Science
Department of Computer Science

Steganography in AVI Files

A THESIS SUBMITTED TO THE COLLEGE OF SCIENCE OF AL-NAHRAIN UNIVERSITY IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

By

Ghessaq Hussein Ali Al – Anbaki

(B.Sc. 2002)

SUPERVISOR

Dr. Taha S. Bashaga

Jamad Al-akher 1426 **July 2005**

الخلاصة

أخفاء المعلومات هو فن أخفاء و أرسال البيانات خلال ناقلات تبدو طبيعية في محاولة لأخفاء وجود البيانات. أعتمد في هذا البحث أخفاء نص أو صورة أو صوت في ملف متعدد الوسائط من نوع **(AVI)**.

في الخطوة الاولى من هذا البحث، يتم فصل **(AVI)** ملف الى جزئين، الفديو و الصوت. جزء الفديو هو عبارة عن سيل من الهياكل الصورية تأخذ كل واحدة على هيئة صور و تخزن في فايل منفصل من نوع **(BMP)**. الخطوة التالية يتم اختيار عدد الهياكل الصورية لغرض استخدامها كغطاء، و يتم قطع المعلومات السرية الى عدد من القطع بناءا على اخفاء كل قطعة في هيكل صوري واحد، ولزيادة الأمنية يتم اختيار الهياكل الصورية المستخدمة للاخفاء بصورة عشوائية.

هناك طريقتين تم استخدامها للاخفاء، الطريقة الاولى (الثنائيات الاقل اهمية) و هي مثال من طرق الاخفاء في المجال المتسلسل، و الطريقة الثانية (نظام التحويل الموجي هار) و هي مثال من طرق الاخفاء في المجال الانتقالي. تم استخدام طريقة الحشر في الثنائي الاقل اهمية **(LSB)** في الثماني لاخفاء ثنائيات البيانات داخل ثمانيات الصور التي من نوع **(BMP)** و من ثم استخدام نفس هذه البيانات في استخراج بيانات الرسالة من الصور. لزيادة امنية النظام التحويل الموجي هار **(Haar Wavelet Transform)** لتقوية الملف المحتوي على البيانات ضد الهجوم.

في جزء الصوت يتم اخفاء معلومات خاصة باستخدام طريقة (الثنائيات الاقل اهمية).

النظام المقترح تم اختباره باستخدام مقياسيين معلوبين قياسية **(MSE, PSNR)**، كل المقاييس المعلوية في اختبار النظام المقترح اظهرت قيم جيدة ل **PSNR**(اكثر من ٤٥ ديسي بيل للفديو و اكثر من ٩٠ ديسي بيل للصوت) و هذه النسبة تزداد بزيادة عدد الهياكل التي تستخدم كغطاء. أما البيانات المسترجعة فكانت هي نفسها البيانات السرية التي تم اخفاءها.

جمهورية العراق
وزارة التعليم العالي
جامعة النهرين
كلية العلوم
قسم علوم الحاسبات

اخفاء المعلومات في الملفات الصوتية الصورية المتداخلة

رسالة
مقدمةالى كلية العلوم في جامعة النهرين كجزء من متطلبات نيل شهادة الماجستير
في علومالحاسوب

مقدمة من قبل

غسق حسين علي العنبكي

(بكلوريوس علوم الحاسبات ٢٠٠٢)

المشرف

د. طه سعدون باشاغا

تموز ٢٠٠٥ **جماد الاخرة ١٤٢٦**

المعلومات الشخصية

الاسم : غسق حسين علي العنبكي

جامعة النهرين، كلية العلوم، قسم علوم الحاسبات

تاريخ المناقشة : ٢٠٠٥/٩/٨

ماجستير علوم حاسبات

عنوان الاطروحة : اخفاء المعلومات في الملفات الصوتية الصورية المتداخلة

المشرف : د. طه سعدون باشاغا

العنوان : محلة ٨٨٣/ زقاق ٤٩/ رقم الدار ٢٠/ حي الجهاد- بغداد- العراق.

رقم الهاتف: ارضي ٥٥٤٦٣٦١
موبايل ٠٧٧٠٢٦٥٩٥٧١

```
[.ShellClassInfo]
LocalizedResourceName=@%SystemRoot%\system32\shell32.dll,-21815
```