Implementation of steganographic software for hiding information in MPEG video files

Artem Romanov¹, Olga Safaryan¹, Larisa Cherckesova¹, and Elena Revyakina¹*

¹Don State Technical University, 1, Gagarin Sq., 344002 Rostov-on-Don, Russia

Abstract. This article is devoted to the implementation of a steganographic software tool for hiding information in MPEG video files. The method of least significant bits was used to hide the data, and error-correcting coding with binary cyclic codes was used to increase the stability of the embedded data. In addition to hiding data, the AES-128 cipher algorithm is applied to the initial data in the algorithm. A software tool has been developed, designed according to the described scheme, which guarantees the impossibility of detecting and reading the transmitted data by third parties, provided that they do not have access to private data. Keywords: computer steganography, data protection, information security, concealment of the fact of information transmission, error-correcting codes.

1 Introduction

At the moment, you can find a huge variety of multimedia files on the Internet, such as digital images, music and videos. All of these files can be used as containers to hide information from prying eyes. Steganography is a science that studies various methods of hiding information in multimedia files, which should remain undetectable both statistically and for human perception. In other words, steganography hides the very fact that some information is hidden in this file.

2 Materials and methods

2.1 Analysis of modern video containers

A video container is a special format for storing digital video. Video data in a container can be stored in two formats: compressed and raw. The video container may also contain audio data encoded in a specific format, subtitles, timing information, and various metadata as a header. The container itself does not define the data encoding format, that is, in theory, video data of any format can be stored in any container. But in practice this is not the case, and each family of codecs has its own container format. Thanks to this, a program that needs to play the contents of a video file can learn about the possibility of playback already

* Corresponding author: revyelena@yandex.ru

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).
from the container itself, without the need to analyze the contents of the container [1]. MPEG-4 video files are used as a container.

MPEG (H.264) is also known as MPEG-4 part 10 and AVC (Advanced video Coding is a standardized digital format for compressing high-definition video at a high bit rate. It is used for more rational use of storage and data transmission devices. The H.264 encoder can reduce the file size of digital video by more than 80% compared to Motion JPEG and 50% compared to MPEG-4 Part 2 without sacrificing image quality. This means much lower bandwidth requirements for transmission and storage space for the video file. Or, on the other hand, the opportunity to get better video quality at the same bit rate. To date, the H.264 format is one of the most progressive and up-to-date compression algorithms [2].

2.2 Color spaces

RGB color space is an additive color model that describes a way to encode colors for color reproduction using three colors: red (red), green (green), blue (blue), which are considered to be primary. The final color is obtained by combining the three components, Figure 1, therefore, using various combinations of these components, any color can be obtained. The most common format is RGB24, which has 8 bits for each color component, and the numeric value of the component is in the range [0; 255] [3].

![Additivity of RGB components.](image)

This color model is widely used in modern technology. For example, liquid crystal displays are made up of cells, each containing three pixels: red, green, and blue. Depending on the image on the screen, each pixel is highlighted with a certain intensity [3]. Due to the additivity of the RGB model from the viewer's side, three pixels of different colors merge into one color, as shown in Figure 2.
YCbCr is a color space consisting of three components: luminance components (Y) and two chrominance components Cb (blue) and Cr (red), Figure 3. This separation is caused by the fact that human vision is more sensitive to brightness than to the color of an object. Therefore, the Cb and Cr chrominance components can be stored at a lower resolution, which can reduce the amount of data stored or transmitted. Therefore, this color space is widely used in digital images and video. Also, this space is often used in steganographic algorithms for working with black and white images, where the luminance channel is used [3].

Convert an image from the RGB color space to the YCbCr space using formula (1) used by the JPEG format. In this formula, the components of the RGB space and the YCbCr space lie in the interval [0; 255] [3].

\[
\begin{align*}
Y &= 0.299R + 0.597G + 0.114B \\
Cb &= 128 - 0.1687R - 0.3313G + 0.5B \\
Cr &= 128 + 0.5R - 0.4187G - 0.0813B
\end{align*}
\]

The reverse transformation from the YCbCr color space to the RGB space can be performed using formula (2), in which the components of both spaces must also be in the interval [0; 255] [3].
2.3 Algorithm of least significant bits (LSB)

LSB - Least Significant Bit - is a method of hiding information in a digital stream, the essence of which is to replace the least significant bits in the container with the bits of a secret message. Due to the fact that replaceable bits do not have much effect on the container, the difference between an empty container and a full one is not perceptible to human perception [4].

LSB method can be applied to images, audio and video. In the case of working with images, the method works as follows. Let's say a 1024x768 image with a color depth of 24 bits (3 bytes per pixel) is used as a container. The color of a pixel is encoded in RGB format, that is, 1 byte is allocated for each color component. The example pixel color is pink, #F4B28F [5,6]. The representation of a single pixel is shown in Figure 4.

Fig. 4. 24 bit pixel representation.

Also assume that the first three bits of the message are 110. Then an example pixel for a filled container would look like Figure 5.

Fig. 5. Filled pixel.

The pixel color has changed to 3/255. Such an insignificant change is completely imperceptible for the human visual system, and, moreover, it may not be displayed at all on low-quality devices [7].

Using this method, the size of an inline message can be up to 1/8 of the size of the container. Also, if necessary, the last 2 or 3 bits can be used, which will affect the quality of the transmitted container, and the message transmission will be more noticeable [8].

LSB method is unstable to such container distortions as compression, rotation, scaling, and to any container destruction attacks. This method can only be used if there is no noise in the channel. Also, the algorithm can only be used in containers to which lossless compression will be applied, since the method itself is applied to raw data before it has been processed [9].

The steganalysis of this method is carried out by anomalous values of the least significant bits.

2.4 Noise-resistant coding

To improve the noise immunity of hidden data, the work uses the Hamming code, the most famous of the first error-correcting codes capable of correcting errors. Works with words in the binary system. An error is guaranteed to be corrected if one bit is changed and detected if two have been changed [10].

\[
\begin{align*}
R &= Y + 1.402(C_r - 128) \\
G &= Y - 0.34414(C_b - 128) - 0.71414(C_r - 128) \\
B &= Y + 1.772(C_b - 128)
\end{align*}
\]
They are named after the American mathematician Richard Hamming who proposed them.

Hamming codes have the property of self-control. This means that errors can be automatically detected during data transfer using this code. The construction of Hamming codes can be done by adding new bits so that the total number of 1s in the representation of any number is (odd) even. Since changing one of the bits of the transmitted code word (regardless of whether this bit is information or check) will change the parity of the total number of ones. Thus, by estimating the number of units in a code word, it is possible to automatically detect the presence of an error [10].

One way to encode and decode data using a Hamming code is to use the generator and parity matrix of the code. In this case, encoding is carried out by multiplying the information word by the generating matrix. Checking the correctness of the code word is carried out by multiplying the code word by the check matrix. The vector obtained as a result of such multiplication is called a syndrome. If the syndrome is nonzero, an error was made in the codeword [11].

One of the most significant properties of these codes is also that these codes make it easy to calculate not only the presence of an error, but also the bit in which she was admitted. So, the syndrome obtained as a result of multiplying the code word by the check matrix is the binary representation of the bit number in which the error was made. This property allows not only to efficiently detect errors, but also to correct them algorithmically quickly and without loss of memory. If an error occurs in an even number of bits, it may go unnoticed [11].

Formulas (3) and (4) represent the generating and checking matrices of codes, respectively.

\[
G = \begin{pmatrix}
1000110 \\
0100101 \\
0010011 \\
0001111
\end{pmatrix}
\]  \hspace{1cm} (3)

\[
H = \begin{pmatrix}
1101100 \\
1011010 \\
0111001
\end{pmatrix}
\]  \hspace{1cm} (4)

The Hamming code is extremely easy to implement and also has a handy error correction property. This property makes it one of the most well-known error-correcting codes [11].

3 Results and discussion

3.1 MPEG Compression

A video stream, at its core, represents a sequence of images. Due to the fact that neighboring images in the stream do not differ much from each other, redundancy can be used in compression.

When compressing video, there are 3 types of frames:

- I-frames form the basis of an MPEG stream, and through them random access to a video fragment is performed. The I-frames themselves are slightly compressed to ensure high image quality.
P-frames are encoded relative to previous frames (I or P) and used as a comparison pattern for the next sequence of P-frames. In this case, a high level of compression is achieved.

B-frames are highly compressed. In order to associate B-frames with a video sequence, it is necessary to use not only the previous but also the next image. B-frames are never used for comparison.

I, P, B frames are combined into groups (GOP - Group Of Pictures) representing the minimum repeating set of consecutive frames, for example: (I 0 B1 _ B2 _ P3 _ B4 _ B5 _ P6 _ B7 _ B8 _ P9 _ B10 _ B11) (I 12 B13 B14 P15 _ B16 B17 P18...) [12].

Frames consist of macroblocks - small fragments of an image with a size of 16 × 16 pixels. The MPEG encoder processor analyzes frames and looks for identical or very close macroblocks by comparing base and subsequent frames. As a result, only the difference data between frames, called the offset vector (vector movement code). Macroblocks that do not contain changes are ignored, and thus the amount of data transferred is significantly reduced. To reduce the impact of errors in data transmission, sequential macroblocks are combined into independent sections (slices). In turn, each macroblock consists of six blocks, four of which carry information about brightness (Y), and the remaining 2 blocks carry information about color difference signals (U/V). Blocks are the basic blocks on which the basic operations of mathematical coding are performed, for example, the discrete cosine transform [12].

It is known that image pixels are correlated with their neighbors because the value of a particular pixel can be predicted from its neighbors. The discrete cosine transform (DCT) reduces this redundancy between pixels. It transforms the original data matrix into a matrix of uncorrelated values (5) using the sums of cosines at different frequencies. The DCT has an inverse transformation (6).

\[
F[u, v] = a(u)a(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} F_{DCT}[x, y] \cos \left( \frac{(2x+1)\pi}{2N} \right) \cos \left( \frac{(2y+1)\pi}{2N} \right) \tag{5}
\]

\[
F_{DCT}[x, y] = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} a(u)a(v)F[x, y] \cos \left( \frac{(2x+1)\pi}{2N} \right) \cos \left( \frac{(2y+1)\pi}{2N} \right) \tag{6}
\]

where \(F_{DCT}\) is the original matrix,

\(F\) is the matrix of DCT coefficients,

\(N\) is the size of matrices \(F\) and \(F_{DCT}\).

\(a(k) = \sqrt{1/N}, \text{if } k=0\sqrt{2} N, \text{if } k>0,\)

\(k = u, \text{or } k = v\)

\(u, v, x, y \in [0, N]\).

The transformation takes place in such a way that the coefficients of the DCT matrix are ordered by frequency. The low-frequency coefficients come first, then the mid-frequency and high-frequency coefficients, Figure 6. The low-frequency coefficients contain the most important information for restoring the original data, and changing them will seriously distort the data after applying the inverse transformation. The high frequency coefficients can be clipped (zeroed out) without much impact on the data after applying the inverse transformation that occurs in the quantization step [13].
YCbCr color space components are taken as data for the matrix. To do this, the Y, Cb, and Cr components are divided into 8x8 or 4x4 blocks, and then DCT is applied to each of the blocks.

At the quantization stage, the matrix of DCT coefficients is transformed into a new matrix with a reduced range of values. The transformation is performed by successively dividing each coefficient of the DCT matrix by the value of the quantization step and rounding the resulting value (7).

\[
F_Q[u, v] = \text{round} \left( \frac{F[u, v]}{Q_{\text{step}}} \right) \tag{7}
\]

where \(F_Q\) is the quantized matrix of DCT coefficients,
\(F\) is the original matrix of DCT coefficients,
\(Q_{\text{step}}\) is the quantization step,
\(u, v \in [0, N]\),
\(N\) is the size of matrices \(F\) and \(F_Q\),
\text{round} is the rounding operation.

Below is an example of quantization of the matrix of DCT coefficients with a step of 5 (8).

\[
\begin{bmatrix}
52.4 & -8.05 & 1.446 & 3.7 \\
12.79 & 4.64 & -9.46 & -0.1 \\
-4.83 & 2.04 & -5.13 & 1.813 \\
1.71 & -3.644 & 2.201 & 0.868 \\
\end{bmatrix}
\quad
\begin{bmatrix}
10 & -2 & 0 & 1 \\
3 & 1 & -2 & 0 \\
-1 & 0 & -1 & 0 \\
0 & -1 & 0 & 0 \\
\end{bmatrix}
\tag{8}
\]

As a result of the rounding operation, part of the data is lost, and when inverse quantization is applied during decoding, the resulting matrix will differ to some extent from the original one [13].
3.2 AES algorithm

In case the information hidden in the video container is discovered and extracted by an unauthor-ized person, so that this information remains unknown to him, the message should be encrypted with a re-liable algorithm.

Advanced Encryption Standard (AES) is a symmetric block cipher adopted by the US government as a standard as a result of a competition between institutes of technology. It replaced the deprecated Da-ta Encryption Standard, which no longer met the requirements of network security, which became more complicated in the 21st century [14].

3.3 Embedding data

The complete information embedding algorithm can be divided into four stages, Figure 7:
1) Converting the original video container.
2) Transformation of embedded information.
3) Embedding data.
4) Reverse conversion to stegovideocontainer.

Before starting embedding, you need to prepare the source video data for embedding. Preparation takes place in five stages:
1) Video is decoded from MPEG-4 format.
2) The resulting video is split into frames in the RGB color space.
3) Each frame is converted from RGB color space to YCbCr space.
4) Each frame is divided into blocks of size $N \times N$, where $N \in [2]$. Smaller values of $N$ allow more information to be embedded, as the number of blocks increases, but also increases the noise.
5) Blocks of the brightness component $Y$ are selected and transformed using DCT into matrices of size $N \times N$.

Before starting the embedding, it is necessary to encode the embedded information using error-correcting codes. Encoding takes place in five stages:
1) The embedded information is converted into a bit sequence.
2) A binary cyclic $(n, k)$ code is chosen to encode information.
3) The bit sequence is divided into segments (information words) of length $k$.
4) Each information word is encoded using the selected binary cyclic $(n, k)$-code.
5) All received code words are combined into one bit sequence.

Once the original video data and embedded data have been converted, embedding can begin. For each DCT coefficient matrix obtained after transforming the original video container, the following steps are performed:
1) The very first bit $b_1$ is taken from the bit sequence obtained at the stage of nested information transformation.
$s > 0$ is selected, on which the stability of the embedded data depends. And the larger this value, the more distorted video frames will be after embedding.
3) The highest frequency coefficient of the DCT matrix is replaced by the resistance value (9).

\[ F[N - 1, N - 1] = \begin{cases} \text{s, if } b = 0 \\ \text{-s, if } b = 1 \end{cases} \]

where \( F \) is the original matrix of DCT coefficients, 
\( N \) is the size of the DCT matrix, 
\( s \) is durability coefficient, 
\( b \) is an embedded bit.

4) The very first bit is removed from the bit sequence. 
The embedding lasts until the bit sequence is empty [15]. 
After embedding the information, all blocks must be assembled into frames and again encoded in the MPEG-4 format.

1) Reverse DCT is applied for each block. 
2) Blocks are combined into frames. 
3) Frames are converted from YCbCr color space to RGB space. 
4) Video is encoded in MPEG-4 format.

### 3.4 Data extraction

The extraction algorithm can be broken down into three steps, Figure 8:

1) Converting the stegovideocontainer. 
2) Data extraction. 
3) Transformation of the extracted data.

Fig. 8. Steps for extracting embedded information.

The first step, transforming the stegovideocontainer, is done in the same way as in embedding. Only in this case it is necessary to know the size of the DCT matrix \( N \), which was used at the embedding stage [15].

After the conversion of stegovideo data, you can start extracting. To extract the data, the length of the embedded message must be known, where the following steps are performed for each DCT coefficient matrix:

1) The highest frequency coefficient of the DCT matrix is checked. If it is positive, then the extracted bit is '0', if it is negative, '1'.
where $b$ is the extracted bit,

$F_M$ is the DCT coefficient matrix with embedded information,

$N$ -is the size of the DCT matrix.

2) The extracted bit is written to a new bit sequence.

Upon completion of the extraction process, it is necessary to decode the data and correct possible errors:

1) For the binary cyclic $(n, k)$-code used when embedding information, a symbol table is built.

2) The bit sequence obtained at the previous data extraction step is divided into segments (code words) of length $n$.

3) For each code word, the error syndrome is checked, and if an error is present, then it is corrected.

4) The decoded information words of length $k$ are combined into a bit sequence.

5) The bit sequence is converted into the original data format [16].

### 3.5 Software implementation

After analyzing the algorithms and methods, the goal was to implement a software tool that, using the steganographic method, hides information in video files. The C# programming language was chosen to write the program code.

The application contains two forms. When the program starts, a window opens prompting the user to select a container file to write a secret message to. After the user has selected the desired file, he can record his message frame by frame into the container, choosing the recording step. In the program window, the user sees a preview of the video file, as well as the frame in which he writes the message. When the user has written a message to frames, he can write his message to the container, saving it on his computer. In the same window there is a button to go to the second window [17].

In the second window, the program prompts the user to extract the message that was previously recorded in the video container. The program offers to view the contents of the message frame by frame, or save the message to a text file in its entirety. The block diagram of the program operation is shown in Figure 9.
The tool consists of 11 classes: Coder, Matrix, Value, Vector, App.prop, Crypto, Program, Stenography, VideoManager, Form1 and Form2. Program class does not contain methods with calculations, it is a class that implements the entrance to our application, that is, it has the main entry point to the application.

A group of classes Coder, Matrix, Value and Vector, combined for convenience in one folder, implement noise-correcting coding to increase the stability of hidden data by Hamming code. Crypto class performs the function of encrypting and decrypting the embedded secret message with the AES algorithm.
Stenography class is used to embed a secret message into the frames of a video container. Using AES encryption and error-correcting coding, the class embeds each bit of the message into the least significant bits of a pixel in the frame. This class also retrieves the previously embedded message from the video container.

VideoManager class contains methods necessary for working with video files using various codecs, including MPEG. The class contains methods such as: direct and inverse discrete cosine transform, opening a video stream for recording a message into frames.

In the main window of the program, Figure 10, the user is shown a preview of the video he has chosen. In the "Write step" column, the default value is "30", that is, the message will be written every thirtieth frame. The user can change this parameter at his own discretion at any stage of the message recording. At the bottom of the main window there is a field in which the user can enter the text of the embedded message.

Fig. 10. Form of writing a message to a video container.

After entering the message, the user presses the "Record" button. After that, the message is injected into the container and saved to a video file, Figure 11.

Fig. 11. Saving the video container.

Upon successful implementation of the message, the corresponding window will be displayed on the screen, Figure 12.
In order to extract information from the video container, the user needs to click on the appropriate button to go to the desired window, Figure 13.

The user then clicks on the "Select Container" button and selects the desired video file. After the user adds a container, the field on the left displays a list of frames in which the message is recorded, as well as the message itself for quick viewing. The field on the right shows the selected frame, and the field below shows the full text of the message in this frame, Figure 14.
The user can save the full text of the embedded message to a text file by clicking on the "Download Message" button, Figures 15, 16.

![Fig. 15. Saving a message to a text file.](image)

![Fig. 16. Message in a text file.](image)

4 Conclusion

The result of the work is a program for steganographic information hiding in MPEG video files. The program implements embedding text information into MPEG video files and extracting embedded information from MPEG video files. The embedding method is based on the least significant bits method. To increase the stability of the embedded data, error-correcting coding with binary cyclic codes was used.

The program can be used to transfer secret data on electronic media and via communication channels.

References