BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

APPLICATIONS FOR MEDIA STEGANOGRAPHY

SEMESTRAL PROJECT
SEMESTRÁLNÍ PROJEKT

AUTHORS **Andrej Krivulčík, Pavla Hlučková, Matěj Miška, Jiří Březina**
AUTOŘI PRÁCE

BRNO 2021

Contents

1 Introduction

This project deals with the creation of an application to use the steganography medium. The aim of the work is to create a standalone application for modifying the medium in order to hide information. The application is created using the Python programming language with the use of libraries for editing multimedia data (`Pillow`, `ffmpeg`, `moviepy`) and to create a graphical user interface (`PyQt5`). The user has the ability to choose the format of the multimedia data and the message that is hidden in the media, along with a password to provide better security of the transmitted data against intentional file corruption and brute-force attacks on message disclosure. Furthermore, the application also supports reverse decryption of information from the media after entering the correct password. The aforementioned multimedia data formats supported by the application are `.png`, `.jpg` image format and `.mp4` video format. After processing, images are saved as `.png` without alpha channel and videos as `.mov`. The application has an intuitive graphical interface for easy media handling.

2 Draft

The goal is to create a user-friendly application that would allow easy hiding of the message in the image and video without being recognizable to the human eye. To make the message more secure, it is not encoded pixel by pixel into the image, but a random number generator is used to determine the encoded pixels. This generator has as „seed“ a custom password that the user enters. This same password must be entered when the message is decoded, otherwise the message will not be readable.

As for the encoding of the message into a video, it is done in a similar way with slight variations. The video must be split into frames before the message is inserted, which then carry the message. In the case of video, the message itself is encoded one character at a time into a different frame, at a different position. After encoding, the video is reassembled and decoding the message is not possible without the encoding password (used to derive other parameters) and re-division into frames.

The application includes a graphical user interface that is clearly divided into sections. This allows for a seamless and simple operation. When an image is loaded for encoding, a preview of the encoded image is displayed, and when the message is hidden, a preview of the encoded image is also displayed for comparison.

3 Theoretical analysis

Steganography is a scientific discipline having the task of transmitting a secret message in the background of a medium through unclassified communication.

The essence of this image manipulation is to hide the message in the RGB pixel values of the medium. To the human eye, this change is imperceptible, but on close inspection, the bit string containing the message can be read from the change in parity of the RGB values.

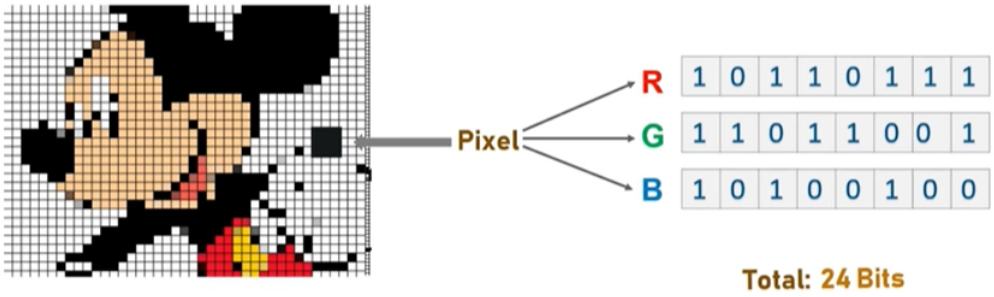

Fig. 1: RGB coordinate value distribution principle.

If video is chosen as the medium for storing the message, it is necessary to decompose the video into the individual frames that make up the video, and then embed the message in one or more frames. These frames must then be reassembled into the pattern of the original video.

4 Solution

The application is built using the Python programming language, using libraries to edit multimedia data (`Pillow`, `ffmpeg`, `moviepy`) and to create a graphical user interface (`PyQt5`).

The encoding process consists of using the `Pillow` library to decompose the image into an array of RGB values, which are then adjusted by a value of 1, thus adjusting the value of the last bit (Least Significant Bit). To the human eye, this change is virtually invisible. Using a random generator, taking as its „seed“ the password entered by the user, the message is split with the indentation created by the length generated. In this way it is achieved that the message is not concentrated in one part of the image, but is spread over the whole image. In this way, it is impossible to extract the message from the image without knowing the password.

The resulting format of the image is `.png`, since no compression occurs during its creation. The input can be any `.png` or `.jpg` file.

A similar process takes place during the decryption process, where a random generator (again, the password here functions as „seed“) is used to generate the offsets between the loading of each RGB component.

The process of video encryption and decryption consists of decomposing the video into individual frames using the `ffmpeg` library, the `moviepy` library extracts the audio, and then the application decomposes the message into individual characters sequentially inserted into the frames in the original sequence (the same as in the message). The actual embedding of the message into the video represents a proof-of-concept, i.e., that it is possible to embed the message into individual arbitrary frames of the video. Then the video is composed again, but this time with the edited frames. The disadvantage of this approach is that the video compositing process cannot include any compression, i.e. the video will be much larger in size than it would normally be after compression. Such a video is saved in `.mov` format.

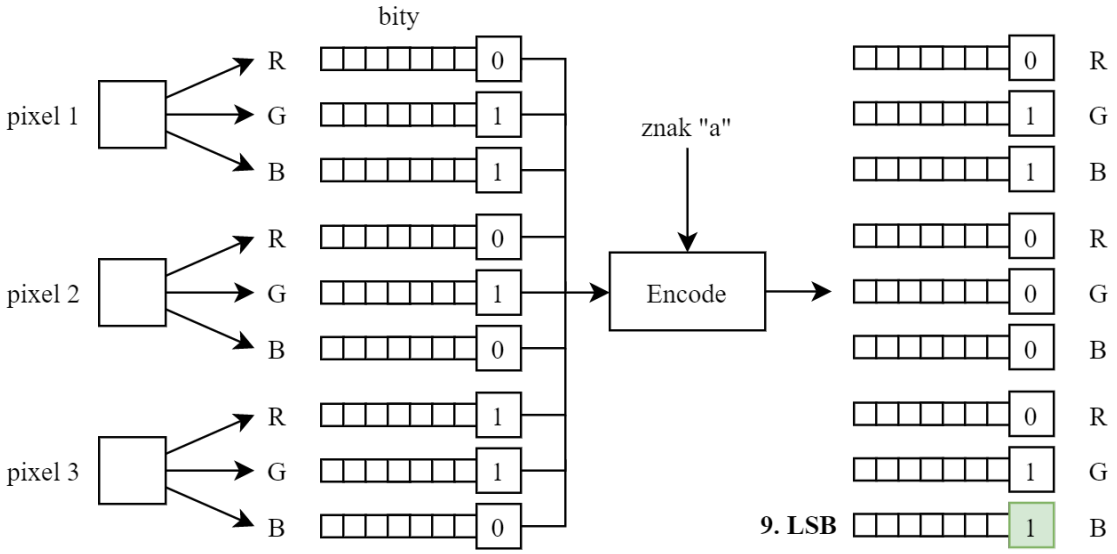

Fig. 2: Principle of encoding a character into pixels.

Fig. 3: Encoding and decoding flowchart of the message in the image.

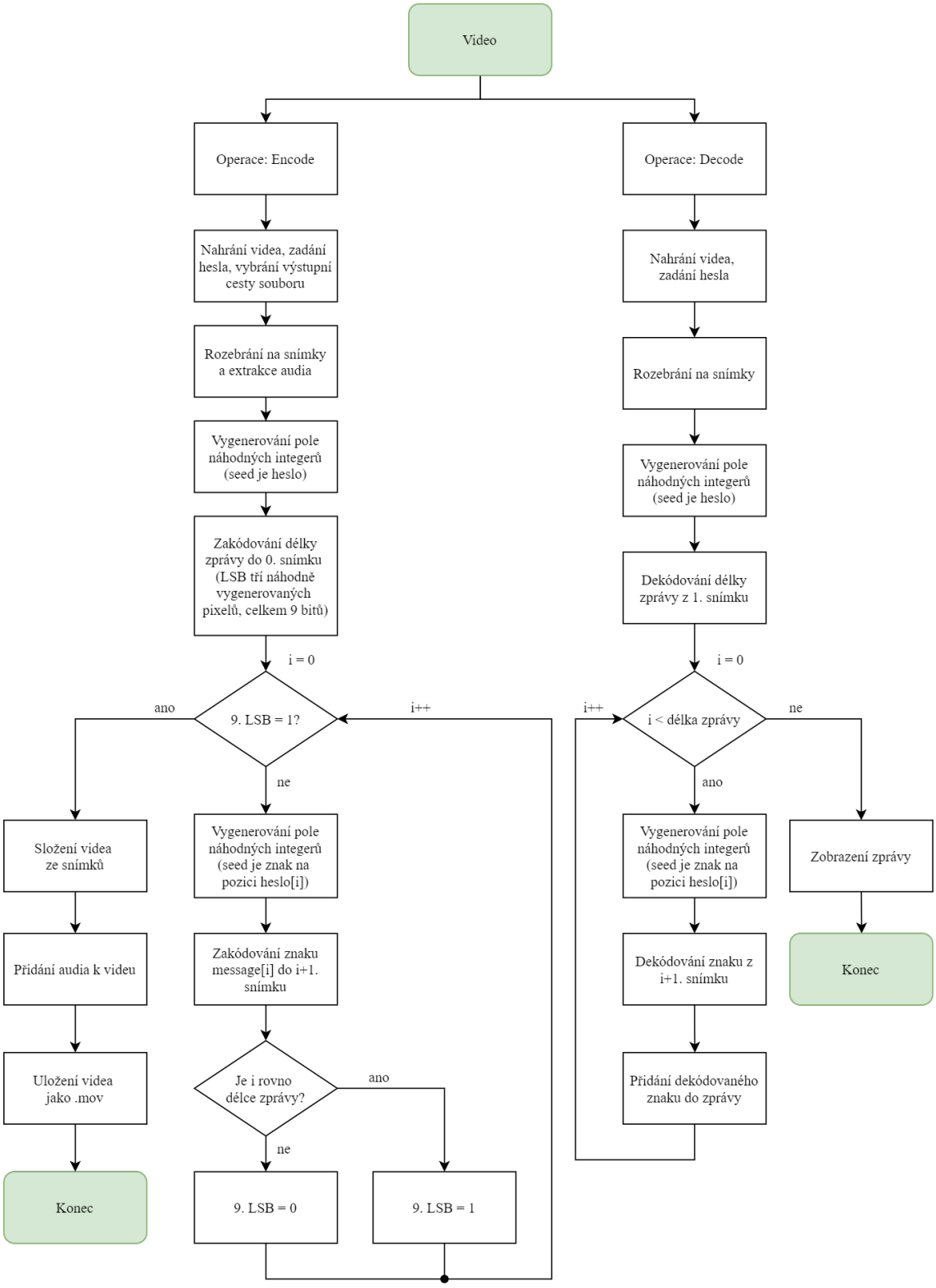

Fig. 4: Video message encoding and decoding flowchart.

5 Application

The application itself is very user friendly. It is clearly divided into several sections, with the first three sections being user accessible and the others being for control only, see Fig. 5.

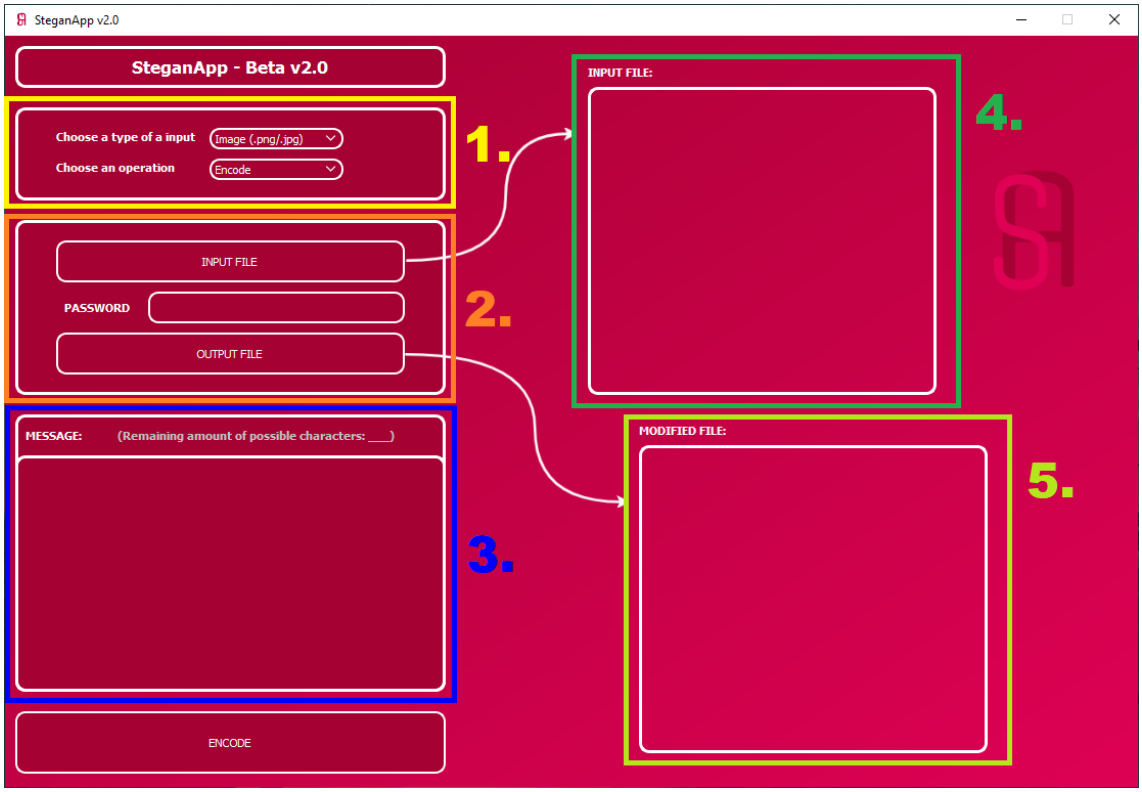

Fig. 5: Basic app interface with sections highlighted.

5.1 Application Usage

a) First section

In first section, **file type** selects *Image (.png/.jpg)* (image files in `PNG` or `JPG` format) or *Video (. mp4)* (video files in `MP4` format) and **operation** itself *Encode* for encryption, or *Decode* if it wants to decrypt an already created file.

b) Second section

Depending on the selection of the operation (encryption or decryption) in the previous section, the user will be shown the available options in the second section. In case the user has selected the *Encode* operation, a total of **two buttons** will be displayed in this section - first button for selecting the source file (*INPUT FILE*),

where the retrieved image will be displayed to the user in **fourth section** including the path to the file. The **second button** is used to name and select the location of the resulting image (*OUTPUT FILE*). Between the buttons is a space for specifying **password** to encrypt the file. For the encryption itself, you still need to write a message in the third section.

If the user had selected the *Decode* option in the previous section, then of course the **not displayed** button for the resulting file will be *OUTPUT FILE* in this section, since no more file is created during decryption and only the encrypted message in the third section is displayed.

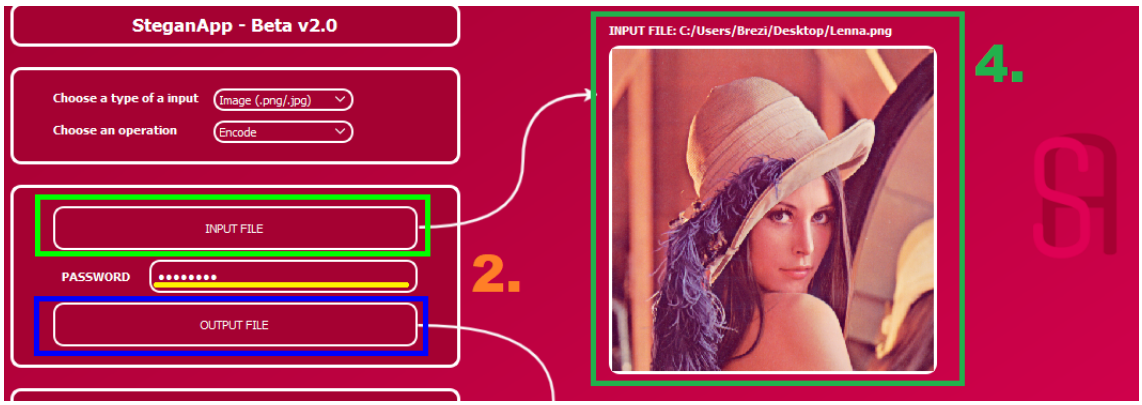

Fig. 6: Detail of the second and fourth section with highlighting of each element.

c) Third section

In **third section**, the user enters the **message** itself into the *MESSAGE:* field that he wants to encode into the image. Above the message field, **number of characters** is displayed to show the maximum number of characters that can be included in the message. Clicking on the **button** *ENCODE* will perform the encryption itself, and then the resulting encrypted image will be displayed in the last **five section**.

In case the user selected the *Decode* operation in the **first** section, the *MESSAGE:* field will display the decrypted message from the selected image (of course, if the user entered the correct password).

Fig. 7: Detail of the third and fifth section with highlighting of each element.

6 Application code documentation

a) `SteganoImage.py`

`convertDataFromString(...):`
convert message to binary

`modifyPixels(...):`
encode message into a triplet of pixels

`encodeToImage(...):`
create, encode and save the message into an image

`decodeFromImage(...):`
decoding message from image

b) `SteganoVideo.py`

`audioFromVideo(...):`
remove audio-track from video

`videoToFrames(...):`
convert video to separate frames

`framesToVideo(...):`
convert frames together with the original audio track back to video

`encodedVideoToFrame(...):`
convert edited video to separate frames

`deleteFrames(...):`
deleting redundant files after the video has been recomposed

`videoEncode(...):`
encoding of the message itself into the video

`videoDecode(...):`
decoding the message from the video

c) `GUI.py`

contains handlers to the GUI and its settings

7 Conclusion and evaluation of work

The application is functional and has a nice user interface. It works for JPG and PNG image formats and MP4 video format. To get the application working, you need to install `ffmpeg` for the Windows command line and add it to the system variable `PATH`, or move the EXE files directly to the System32 folder. Next, you need to install the aforementioned `Pillow`, `moviepy`, `PyQt5` and `ffmpeg-python` libraries.

8 Literatura

[1] *Steganography*. [online]. [cit. 2021-04-26]. Dostupné z: `https://en.wikipedia.org/wiki/Steganography`

[2] *Steganography in video*. [online]. [cit. 2021-04-26]. Dostupné z: `https://bit.ly/32OGkG9`

[3] *Python-ffmpeg*. [online]. [cit. 2021-04-26]. Dostupné z: `https://kkroening.github.io/ffmpeg-python/`

[4] *MoviePy*. [online]. [cit. 2021-04-26]. Dostupné z: `https://zulko.github.io/moviepy/`

[5] *Pillow*. [online]. [cit. 2021-04-26]. Dostupné z: `https://pillow.readthedocs.io/en/stable/`

[6] *PyQt5*. [online]. [cit. 2021-04-26]. Dostupné z: `https://pypi.org/project/PyQt5/`

[7] *Qt Designer Manual*. [online]. [cit. 2021-04-26]. Dostupné z: `https://doc.qt.io/qt-5/qtdesigner-manual.html`

[8] *RGB distribution*. [online]. [cit. 2021-04-26]. Dostupné z: `https://bit.ly/3aFssSR`