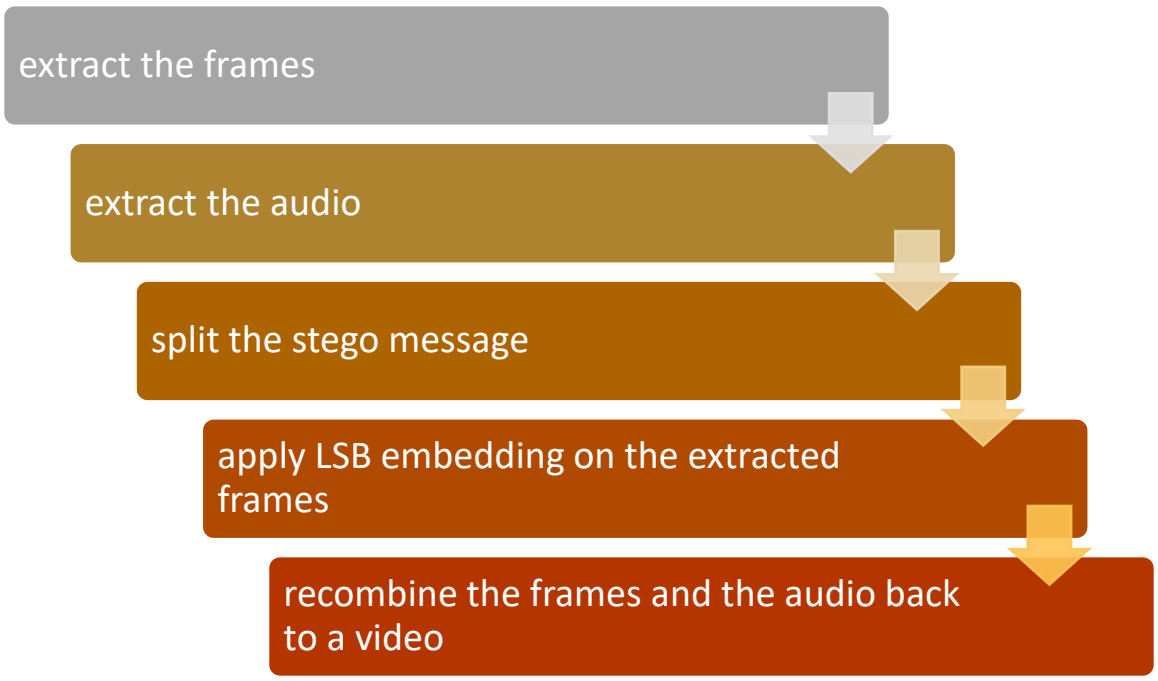
LSB Video Steganography

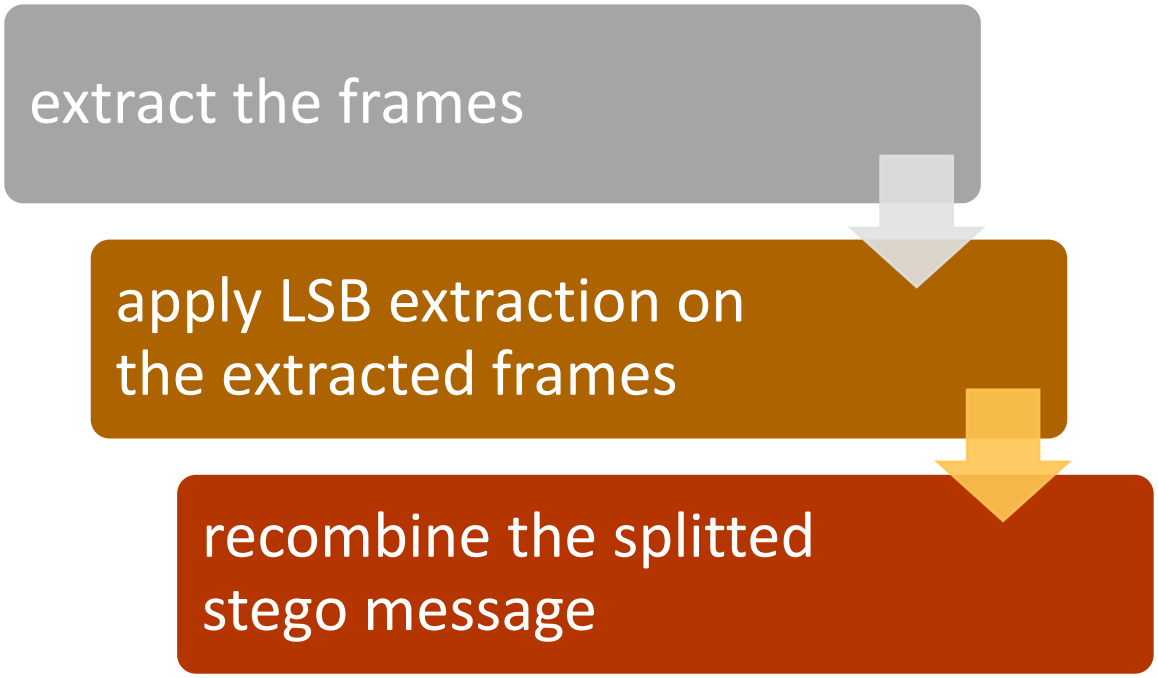
Video steganography is the art of concealing secret information within video files to transmit covert messages without arousing suspicion. Employing techniques such as LSB substitution, frequency domain manipulation, and spread spectrum methods, video steganography enables the seamless integration of hidden data into the visual and temporal aspects of a video. This clandestine practice is often employed for secure communication, digital watermarking, or copyright protection, leveraging the imperceptibility of alterations to individual frames. As a subset of steganography, video steganography plays a crucial role in safeguarding information while embedded within the seemingly innocuous medium of video content.

LSB (Least Significant Bit) video steganography is a covert communication technique that exploits the imperceptibility of small changes in the least significant bits of video frames to conceal hidden information. In this method, binary data is inserted by replacing the least significant bits of pixel values in the video, allowing for the embedding of additional information without significantly altering the visual quality of the content. LSB steganography is widely employed due to its simplicity and effectiveness, making it a popular choice for hiding data within the binary structure of video files. The concealed information can be later extracted by decoding the altered LSBs, revealing the hidden message without attracting attention.

The steps to be taken to implement the embedding of video LSB can be shown in the following visual:



The steps to be taken to implement the extraction of video LSB can be shown in the following visual:



A video file format is a type of file format for storing digital video data on a computer system. Video is almost always stored using lossy compression to reduce the file size.

A video file normally consists of a container (e.g. in the Matroska format) containing visual (video without audio) data in a video coding format (e.g. VP9) alongside audio data in an audio coding format (e.g. Opus). The container can also contain synchronization information, subtitles, and metadata such as title. A standardized (or in some cases de facto standard) video file type such as .webm is a profile specified by a restriction on which container format and which video and audio compression formats are allowed.

The coded video and audio inside a video file container (i.e. not headers, footers, and metadata) is called the essence. A program (or hardware) which can decode compressed video or audio is called a codec; playing or encoding a video file will sometimes require the user to install a codec library corresponding to the type of video and audio coding used in the file.

The main challenge that LSB faces is the modification of the bits that may happen during the compression of the file.

There are two types of video codecs: lossy (compression) and lossless (no compression). The lossy codec is the biggest enemy to LSB embedding. On the other hand, lossless codecs are VERY secure in LSB embedding.

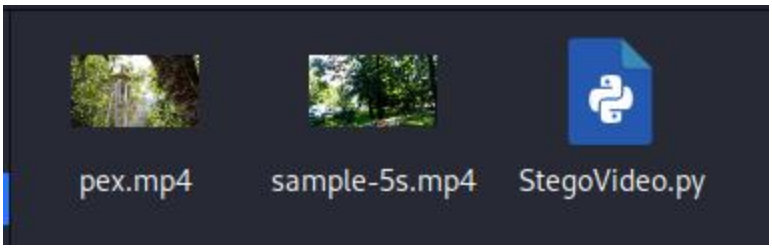
When we want to embed, we need to choose a video where the file container supports lossless video codecs, such as ffv1, H.264 Lossless, H.265 Lossless, Motion JPEG Lossless, Apple Animation Quicktime RLE, Autodesk Animator Codec, and many more. The file containers that support lossless codecs are MKV (Matroska), and AVI (Audio Video Interleave).

I chose the video container, and the codecs as follows:

1. AVI (Audio Video Interleave), because it supports lossless codecs and the audio and video are on different streams, so that even if the audio stream is harmed; the secret message remains.
2. ffv1 (FF Video 1), because it is a lossless codec, so it can carry secret data.

Code implementation can be found in the accompanying files.

Files:



Run results:

embedding

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python StegoVideo.py embed sample-5s.mp4 --message "secret message hidden deep inside >:3"
./tmp  directory is created
Frames =  171
Extracting frames . . .
Extracting frames: DONE.
[INFO] frame ./tmp/0.png holds secr
[INFO] frame ./tmp/1.png holds et m
[INFO] frame ./tmp/2.png holds essa
[INFO] frame ./tmp/3.png holds ge h
[INFO] frame ./tmp/4.png holds idde
[INFO] frame ./tmp/5.png holds n de
[INFO] frame ./tmp/6.png holds ep i
[INFO] frame ./tmp/7.png holds nsid
[INFO] frame ./tmp/8.png holds e >:
[INFO] frame ./tmp/9.png holds 3
[INFO] ./tmp files are cleaned up
```

extraction

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python StegoVideo.py extract stego_video.avi
Starting the extraction . . .
./tmp  directory is created
Frames =  171
Extracting frames . . .
Extracting frames: DONE.
frame  ./tmp/0.png contains:  secr
frame  ./tmp/1.png contains:  et m
frame  ./tmp/2.png contains:  essa
frame  ./tmp/3.png contains:  ge h
frame  ./tmp/4.png contains:  idde
frame  ./tmp/5.png contains:  n de
frame  ./tmp/6.png contains:  ep i
frame  ./tmp/7.png contains:  nsid
frame  ./tmp/8.png contains:  e >:
frame  ./tmp/9.png contains:  3
secret message=  secret message hidden deep inside >:3
[INFO] ./tmp files are cleaned up
```

SSIM

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python StegoVideo.py compare sample-5s.mp4 --stego_path stego_video.avi
Starting the comparison. . .
./tmp1  directory is created
Frames =  171
Extracting frames . . .
Extracting frames: DONE.
./tmp2  directory is created
Frames =  171
Extracting frames . . .
Extracting frames: DONE.
Completed  10  SSIM comparisons.
Completed  20  SSIM comparisons.
Completed  30  SSIM comparisons.
Completed  40  SSIM comparisons.
Completed  50  SSIM comparisons.
Completed  60  SSIM comparisons.
Completed  70  SSIM comparisons.
Completed  80  SSIM comparisons.
Completed  90  SSIM comparisons.
Completed  100  SSIM comparisons.
Completed  110  SSIM comparisons.
Completed  120  SSIM comparisons.
Completed  130  SSIM comparisons.
Completed  140  SSIM comparisons.
Completed  150  SSIM comparisons.
Completed  160  SSIM comparisons.
Completed  170  SSIM comparisons.
SSIM Score:  1.0
[INFO] ./tmp1 files are cleaned up
[INFO] ./tmp2 files are cleaned up
```

Notes:

- the code and input/output videos are provided in the other files.
- Run the code in linux.
- This time of running may vary depending on the file size and quality.
- The input file can be of any video type.
- The output file will always be avi.
- SSIM score is rounded to 2 decimals so small changes (like the output above) may still give a perfect score of 1.0.
- SSIM here is the average SSIM of all frames.
- You can try to compare the sample-5s with pex file to confirm SSIM is working.
- You need to use a video player that supports ffv1 like VLC media player.

DISCLAIMER:

- **The code is provided for educational purposes only. It is intended to help individuals learn about coding, programming, and related concepts. The use of this code for any malicious or unauthorized activities is strictly prohibited. The author is not responsible for any misuse or damage caused by this code. Use it responsibly and within the bounds of ethical and legal guidelines.**
- **Do not share my work without my permission.**
- **If you want permission, please email me on mailto:azooztbaishat@gmail.com**

REFERENCES:

- llopen-sourcell/Video-Steganography: python program to hide texts inside a video using LSB transformation (github.com) on GitHub.
- FFV1 - Wikipedia
- Lossless Codecs and Lossy Codecs: What does that mean? (promax.com)
- ChatGPT.