

# **Heritage Institute of Technology**

*(An Autonomous Institute)*

## **Department of Electronics and Communication Engineering**



----- Project Report -----

### **Digital Right Management System with Visual Morse Signal Steganography**

*Affiliated*

*to*

***Maulana Abul Kalam Azad University of Technology***

***(Formerly WBUT), 2020***

| <b>Name</b>             | <b>Roll No</b> | <b>Registration No</b> |
|-------------------------|----------------|------------------------|
| Rajarshi Dey            | 12617003109    | 171260110354           |
| Saptarshi Roy Chowdhury | 12617003127    | 171260110372           |
| Rahit Saha              | 12617003107    | 171260110352           |
| Anuran Bose             | 12617003042    | 171260110289           |

**Heritage Institute of Technology**  
(An Autonomous Institute)  
**Department of**  
**Electronics and Communication Engineering**



*This is to certify that the project report*

-----**TITLE OF THE PROJECT**-----  
*Has been successfully completed by*

| <b>Name</b>             | <b>Roll No</b> | <b>Registration No</b> |
|-------------------------|----------------|------------------------|
| Rajarshi Dey            | 12617003109    | 171260110354           |
| Saptarshi Roy Chowdhury | 12617003127    | 171260110372           |
| Rahit Saha              | 12617003107    | 171260110352           |
| Anuran Bose             | 12617003042    | 171260110289           |

*In partial fulfillment for the award of the degree in*  
*Bachelor of Technology*  
**In**  
*Electronics and Communication Engineering*  
*Maulana Abul Kalam Azad University of Technology*  
*(Formerly WBUT), 2020*

*Under the Supervision of*  
*Prof. Anindya Sen*

(Name of the Project Guide) ----- **Anindya Sen**  
(Designation of Guide) ----- Professor

Dept. of Electronic and Communication Engineering  
Heritage Institute of Technology, Kolkata-700107.



## *Certificate of Recommendation*

This is to certify that the Thesis entitled “***Digital Right Management system using Visual Morse Code Steganography***” submitted by **Rahit Saha, Anuran Bose, Rajarshi Dey, Saptarshi Roy Chowdhury** under the supervision of ***Prof. Anindya Sen***(Professor, Dept. of ECE, HITK), has been prepared according to the regulations of **B.Tech. Degree in Electronics and Communication Engineering Department**, awarded by **Maulana Abul Kalam Azad University of Technology (Formerly WBUT)** and he/she has fulfilled the requirements for submission of thesis report and that neither his/her thesis report has been submitted for any degree/diploma or any other academic award anywhere before.

.....  
-----**Prof.---****Name of Project Guide---**  
(Designation, Dept of ECE, HITK)  
***Project Supervisor***

.....  
**Prof. Prabir Banerjee**  
(HOD, Dept of ECE, HITK)

# **Heritage Institute of Technology**

*(An Autonomous Institute)*

*Affiliated to*

## **Maulana Abul Kalam Azad University of Technology**

(Formerly WBUT)



### ***Certificate of Approval\****

The foregoing thesis report is hereby approved as a creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned don't necessarily endorse or approve any statement made opinion expressed or conclusion drawn therein but approve the project report only for the purpose for which it is submitted.

*Signature of the Examiners:*

1.....

2.....

3.....

*\*Only in the case the thesis report is approved*

## **ABSTRACT**

Time changes, so does the mode of entertainment in our life. From the longest silver screens and theatres, to the 5-inch smartphone LED graphics, technology inhabits the human psychology and makes entertainment more essential than it ever was. Budgets increase, so does the infamous peer to peer nodes, unauthorized theft, piracy - shallowing the information security margin. Losses of the best entertainment projects becomes inevitable, and hence the need for copyright security knocks the doors of the total IT industry.

## ACKNOWLEDGEMENTS

It is our privilege to express our sincerest regards to our project coordinator, Prof. Anindya Sen, for his valuable inputs, able guidance, encouragement, whole- hearted cooperation and constructive criticism throughout the duration of our project.

We deeply express our sincere thanks to our Head of Department Dr. Prof. Prabir Banerjee and Departmental coordinator Shounak Dasgupta for encouraging and allowing us to present the project on the topic "**Digital rights management using Visual Morse Signal Steganography**" at our department premises for the partial fulfillment of the requirements leading to the award of B-Tech degree.

We take this opportunity to thank all our lecturers who have directly or indirectly helped our project. We pay our respects and love to our parents and all other family members and friends for their love and encouragement throughout our career. Last but not the least we express our thanks to our friends for their continuous cooperation and support throughout entire four years.

## Index

### Contents

|                 |   |    |
|-----------------|---|----|
| Abstract        |   | 05 |
| Acknowledgement |   | 06 |
| Chapter 1       | Introduction                              | 08 |
|                 | Objective                                 | 09 |
| 1.0             | Purpose                                   | 10 |
| 1.1             | Digital Right Managements & Steganography | 12 |
| 1.1.0           | Steganography                             | 12 |
| 1.1.1           | Background and related work               | 13 |
| 1.1.2           | Cryptographic DRM schemes                 | 14 |
| 1.1.3           | Streaming DRM Platforms                   | 15 |
| Chapter 2       | Materials & Method                        | 18 |
| 2.1             | Planning of Experiment                    | 19 |
| 2.1.0           | Details of Data                           | 20 |
| 2.1.1           | Morse Code                                | 21 |
| 2.2             | Algorithm                                 | 23 |
| 2.2.1           | Logic Details                             | 24 |
| Chapter 3       | Experiment Result                         | 33 |
| 3.1             | Result                                    | 34 |
| Chapter 4       | Analysis & Conclusion                     | 35 |
| 4.1             | Analysis                                  | 36 |
| 4.2             | Conclusion                                | 37 |
| 4.3             | Future possible R&Ds                      | 38 |
| References      |   | 39 |

# Chapter 1: Introduction



## Objective

In this project we are going to get an idea about a basic way where we can embed a piece of data in a video in a way that the piece of data can be extracted even if we screen record the video. In this way a streaming service can inject a piece of unique metadata into the videos while streaming it into its consumer. If the respective user or individual extracts the video and sell or make it available in some third-party server or node without consent, the streaming service can take steps to eliminate the user license of the individual and take other necessary steps against the alleged piracy. And like that the IT community can fight for Digital Rights managements.

## 1.0 – Purpose

### Piracy

Online piracy is the practice of downloading and distributing copyrighted content digitally without permission, such as music, video or software. The principle behind piracy has predated the creation of the Internet, but its online popularity arose alongside the internet. Despite its explicit illegality in many developed countries, online piracy is still widely practiced, due to both the ease with which it can be done and the often defensible ethics behind it.

### Piracy Rate

Global online piracy has been on the rise for years now. Consumers are continuously looking for new ways to stream or download their favorite movies, TV shows, or songs for free. Therefore, it's not surprising that some of them try illegal methods to achieve their goal.

- Pirated videos get over 230 billion views a year.
- The USA has the most visits to piracy sites.
- 71,000 jobs are lost in the USA due to online piracy
- 16% of online software on personal computers in the USA is unlicensed
- 70% of online users find nothing wrong with online piracy.
- In May 2019, Superhero movies were the most pirated content.
- According to a 2017 survey, \$315 million in book sales were lost due to eBook piracy.

### Copyright

Copyright is one tool to prevent the intellectual property of a person from being pirated. It is the legal right granted to a creator of any intellectual property to be able to reproduce and redistribute his work, at his discretion. Although back then, and even today, copyright doesn't exactly prevent piracy, it does protect the legal interests of the party negatively affected and prescribe legal consequences for the perpetrator, in the event that copyright infringement (piracy) does occur. Copyright holders routinely invoke legal and technological measures to prevent and penalize copyright infringement.

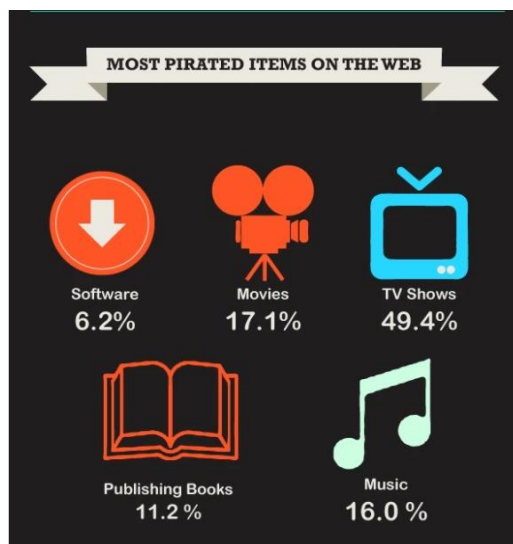
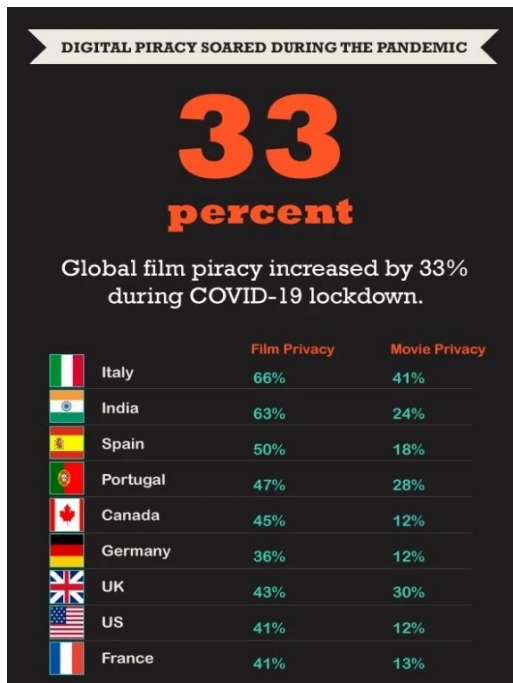
The Copyright Act, 1957 handles protection of copyrighted material via classification of the same into two categories of rights, those being –

- **Economic Rights:** The scope of this Act falls under originally conceptualized work including literary works, dramatic works, musical works, artistic works, cinematograph films, and sound recordings. The owners of these intellectual properties and works are given exclusive rights which they can exercise when it comes to the reproduction and distribution of these works, and to have a share in the profit of any sales of the product made by a licensed third-party.
- **Moral Rights:** Section 57 of the Act splits moral rights into two basic rights, right of paternity and right of integrity. The former enables the original creator of the intellectual property to be able to claim ownership of it and prevent any others from claiming ownership. The latter enables the creator to restrict any and all 'distortion, mutilation or other alterations of his work, or any other action in relation to said work' which may damage his reputation.

### Charges for piracy

Since the crime of piracy is not limited to only the movie industry, the punishment specified above isn't the only one dealt to pirates. It varies with the industry in which they are committing an act of piracy. The most notable forms of punishment are covered in the provisions of the Copyright Act, 1957 and Information Technology Act, 2000. The punishments specified are as follows-

- **Copyright Act:** If a person uses a pirated computer program, or a program that has been manufactured or acquired through copyright infringement, on any computer device, he shall be liable for imprisonment no less than 7 days, extending up to 3 years, and a fine no less than Rs. 50 thousand, which may be extended up to Rs. 3 lakhs.
- **IT Act:** If a person gains access to a computer, a network of computers, or computer systems, then proceeds to view, copy and extract the data present on the computer, either through digital means or through a removable storage medium (pen drive or hard disk), without prior authorization from the owner of the computer, he is liable to pay damages as compensation which can go up to a sum of Rs. 1 crore. Any person who downloads said stolen data will also be liable for the same amount.



## **1.1 - Digital Right Managements & Steganography**

### **Digital rights management**

Digital rights management (DRM) means protection of copyrighted works by various means to control or prevent digital copies from being shared over computer networks or telecommunications networks. Digital rights management, or DRM, is the application of systems and technologies to protect digital media against copyright infringement. This approach aims to protect the rights of original copyright holders and minimize the unauthorized redistribution of digital media and proprietary software.

In a way, digital rights management allows publishers or authors to control what paying users can do with their works. For companies, implementing digital rights management systems or processes can help to prevent users from accessing or using certain assets, allowing the organization to avoid legal issues that arise from unauthorized use. Today, DRM is playing a growing role in data security.

With the rise of peer-to-peer file exchange services such as torrent sites, online piracy has been the bane of copyrighted material. DRM technologies do not catch those who engage in piracy. Instead, they make it impossible to steal or share the content in the first place.

### **1.1.0 - Steganography**

The practice of hiding a secret message inside of (or even on top of) something that is not secret is called steganography. That something can be just about anything you want. These days, there are many examples of steganography that involve embedding a secret piece of text inside of a picture or hiding a secret message or script inside of a Word or Excel document.

The purpose of steganography is to conceal and deceive. It is a form of covert communication and can involve the use of any medium to hide messages. It's not a form of cryptography, because it doesn't involve scrambling data or using a key. Instead, it is a form of data hiding and can be executed in clever ways.

There are a number of apps that can be used for steganography, including Steghide, Xao, Stegais and Concealment.

### **Artificial Intelligence and Steganography**

Nowadays, increasingly, we're seeing AI uses of various tactics, including steganography, to hide information. AI implementations have even been able to modify steganographic techniques so that attacks can't be easily discovered.

### **Detecting Steganography**

Security analysts work to identify the tactics, techniques and procedures (TTPs) of attackers and pen testers. Over the years, they have identified typical signatures that steganographic applications use. This is why antivirus applications, for example, can identify typical moves made by steganographic applications.

Therefore, pen testers and attackers morph and modify their procedures to thwart detection. Attackers constantly modify tools and techniques, while security analysts constantly look for new signatures and methods.

### 1.1.1 - Background and related work

Over the last several decades, there has been an arms race between content owners, wishing to restrict the use of their content, and content consumers, who wish to use such content in an unrestricted way. New Digital Rights Management techniques are created on a regular basis, and new workarounds are quickly found to counter them.

#### **Media companies:**

In the media industry, DRM technology helps musicians, movie professionals, authors, and other creators to combat unauthorized use of their content. If people can freely share this type of content, the artists and producers will struggle to earn an income for their creations.

In the late 1990s, Napster rose to infamy through its peer-to-peer file sharing platform, making it easy for anyone to download pirate music. Today, the Apple iTunes Music Store uses DRM technology to ensure people can only play music on authorized devices or read iBooks on Apple devices.

#### **Technology companies:**

A report from DataProt found that 57% of computer users admit they have pirated software in the past. In the age of software-as-a-service (SaaS), it's imperative for technology companies to protect their valuable software products from piracy.

Due to this obvious threat, digital rights management is an essential, omnipresent facet of software in 2021. For instance, Microsoft users must acquire a personal user license and input their unique key before installing any Windows or Office software on their personal computer.

#### **Enterprise:**

Enterprise digital rights management (EDRM) has grown into its own market, with Gartner projected EDRM to be worth over \$330M by the end of 2026. Typically, enterprises rely on DRM content to protect critical data, especially during product design documents and M&A (merger and acquisition) plans.

Through good digital rights management software, an enterprise can rapidly deploy new campaigns and product concepts or expedite new market adoption and growth, all while staying compliant with regulatory laws.

#### **Agencies:**

Agencies take responsibility for their client's creations and campaigns, so there is a high expectation for data security and privacy. Whether it's a product launch, rebrand, or new sales page, you need to be sure you can guarantee asset security.

Many agencies use DRM content to streamline workflows, as they can approve items directly in the platform, and enhance the level of brand consistency across the campaign, even when working with in-house personnel and outsourced contractors.

DRM schemes can generally be split into two classes: non-cryptographic DRM schemes and cryptographic DRM schemes. The former relies on verifying that the user is authorized to use the protected content by somehow utilizing a physical aspect of this content. Of course, this requires that the content ships with something like a manual, disk, or hardware dongle to use for verification. With the advent of digital distribution for software and multimedia, non-cryptographic DRM schemes have fallen in popularity. On the other hand, cryptographic DRM schemes work by cryptographically verifying that the user attempting to access the content is authorized to do so. This approach is usable for digital distribution of content, and is the paradigm according to which modern DRM schemes are developed.

### 1.1.2 - Cryptographic DRM schemes

One of the early examples of cryptographic DRM techniques was the DVD Content Scramble System. CSS is an encryption scheme to prevent DVDs from being played on unauthorized devices. It functioned by assigning a limited number of keys to manufacturers of DVD playing devices. These keys would then be used to decrypt the key chain of a given DVD and play back the video. CSS was broken in 1999 through cryptanalysis by a group of security researchers including Jon Lech Johansen (afterwards known as DVD Jon). This was done by reverse engineering a software DVD player to identify the encryption algorithm. CSS was a forerunner of the type of copy protection that Movie Stealer was created to analyze. While DRM schemes have since evolved to be more flexible, the basic premise remains the same: content is shipped in an encrypted form (whether through physical media or as a download), and is decrypted by an authorized player.

#### **Hardware-based DRM:**

Hardware-based DRM has been around since the early days of copy protection. Early examples of this class of approaches are copy-protection dongles shipped with software. Software protected by such dongles does not run without the presence of the dongle, and duplication of the dongle is infeasible. While early dongles simply contained static information that would be checked in software, modern dongles are complex cryptographic co-processors that actually carry out operations, such as decrypting the program code, on behalf of the protected program. A specific adaptation of this into the realm of multimedia is HDCP, a link protection scheme which moves the decryption of media content outside of the computer. In a perfect implementation of this scheme, all content handled by the computer is always encrypted, and the decryption occurs in the media playback hardware (such as the monitor) itself. The main drawbacks for this scheme are, HDCP does not integrate seamlessly with the encryption used in the media streaming services of which we are aware. Encrypted content streamed from these services must be decrypted before being re-encrypted with HDCP. While some media devices exist that can handle this step in dedicated hardware, thus disallowing any access to the unencrypted stream, general purpose consumer devices are not among them. This means that, even in the presence of HDCP, Movie Stealer can intercept the protected content on such devices while it is unencrypted. Finally, HDCP has been irrevocably broken with the leak of the HDCP master key. Hardware based DRM schemes like HDCP are very hard to patch because they need to work on many devices that are not easily upgradeable. While the upgradeability of these devices might be improved in the future, there is currently no clear solution to this issue.

## 1.1.3 - Streaming DRM Platforms

### 1.1.3.1 - Microsoft PlayReady (used by Netflix)

Microsoft's PlayReady DRM, as implemented in its Silverlight streaming platform, which is used most prominently by Netflix, is a cross-platform content protection mechanism. PlayReady supports individualization, meaning that the media is encrypted with a content key, which is then encrypted with different keys for every user. Every time a user streams content on silverlight, PlayReady provides an individualized license, ensuring that the content key can be decrypted and protected content viewed only by the intended recipient. The process to play back PlayReady-protected media using silverlight comprises several steps.

#### Metadata

To initialize playback, the Silverlight client requests metadata from the media server provider (such as Netflix). This metadata is a file that contains available resolutions and bit rates for the content, whether the payload is encrypted or not, the domain name of the license server, and the expiration time of the request.

#### License

If the metadata specifies that the payload is encrypted, the Silverlight client must acquire the license (containing the decryption key) from the license server, which is specified in the metadata. When a client sends the license request to the license server, the license server responds with the Individualized Black Box (IBX). The IBX is a custom, easily-upgradeable, and highly-obfuscated DLL that can be customized by individual content providers. Using the IBX, the client generates an individualized request to the license server. The license server verifies this request and responds with a license. The client uses the IBX to decrypt the license and extract the content key, which is a 128-bit AES key.

#### Data

Having acquired the license, the client can now play back the protected content. This content takes the form of a fragmented MPEG-4 file transferred from the service provider. The protection works by encrypting the media stream data, while leaving the headers and stream metadata unencrypted. The data is encrypted using AES and is decrypted using the key acquired from the license server.

Performance - PlayReady has several performance requirements. To begin with, as with any network service, the client must be able to communicate with the server without letting the connection time out. Additionally, as a security measure against piracy, the IBX and corresponding license request have an expiration time, and the license will stop working after this timeout has elapsed. Finally, the media player (Netflix) itself has a minimum performance threshold, below which it will stop processing the stream and display an error. A successful online analysis of a PlayReady-protected media player must have a low-enough overhead to allow the player to meet these performance obligations.

### **1.1.3.2 - RTMPE**

(used by Adobe's Flash streaming platforms such as Amazon Prime Video and Hulu)

RTMPE is a lightweight link protection mechanism developed by Adobe on top of the Real Time Messaging Protocol (RTMP). The addition to RTMP is a simple encryption layer.

RTMP is a TCP-based protocol which maintains persistent connections and allows low-latency communication. To deliver streams smoothly and transmit as much information as possible, it splits streams into fragments, and their size is negotiated dynamically between the client and server. Sometimes, it is kept unchanged; the default fragment sizes are 64 bytes for audio data and 128 bytes for video data and most other data types. Fragments from different streams may then be interleaved, and multiplexed over a single connection. With longer data chunks, the protocol thus carries only a one-byte header per fragment, so incurring very little overhead. However, in practice, individual fragments are not typically interleaved. Instead, the interleaving and multiplexing is done at the packet level, with RTMP packets across several different active channels being interleaved in such a way as to ensure that each channel meets its bandwidth, latency, and other quality-of-service requirements. Packets interleaved in this fashion are treated as indivisible, and are not interleaved on the fragment level.

The RTMP defines several virtual channels on which packets may be sent and received, and which operate independently of each other. For example, there is a channel for handling RPC requests and responses, a channel for video stream data, a channel for audio stream data, a channel for out-of-band control messages (fragment size negotiation, etc.), and so on. During a typical RTMP session, several channels may be active simultaneously at any given time. When RTMP data is encoded, a packet header is generated. The packet header specifies, amongst other matters, the ID of the channel on which it is to be sent, a timestamp of when it was generated (if necessary), and the size of the packet's payload. This header is then followed by the actual payload content of the packet, which is fragmented according to the currently agreed-upon fragment size before it is sent over the connection. The packet header itself is never fragmented, and its size does not count towards the data in the packet's first fragment. In other words, only the actual packet payload (the media data) is subject to fragmentation.

At a higher level, the RTMP encapsulates MP3 or AAC audio and FLV1 video multimedia streams, and can make remote procedure calls (RPCs) using the Action Message Format. Any RPC services required are made asynchronously, using a single client/server request/response model, such that real-time communication is not required.

#### **Encryption layer**

RTMPE generates a stream key using a Diffie-Hellman key exchange. Once this key is agreed upon, the entire communication stream is encrypted using RC4. No extra encryption is done on the media itself.

Performance – Any online analyzer running against RTMPE must be fast enough to allow the processing of the data stream without dropping the connection.



### 1.1.3.3 - Spotify

Spotify implements a custom protection scheme to prevent duplication of their content. This scheme was reverse engineered by the Despotify Project in their attempt to create an interoperable client. The scheme uses a stream cipher to protect its communication, and, in addition, it encrypts each individual song.

Stream cipher - The Spotify client performs a key exchange with the server to create a key to be used for the remainder of the session. After the key is generated, the session is encrypted using a Shannon stream cipher.

Song encryption - Individual music files sent by Spotify in the encrypted stream are themselves encrypted with AES. The keys to this encryption are sent in the stream along with the music files. Upon receipt of a music file and its corresponding key, the Spotify client decrypts the file for playback. For offline playback, Spotify can cache this data.

#### Performance

An online analysis of Spotify must be fast enough to process the data stream without dropping the connection. Additionally, if the Spotify client runs too slowly, it will mistakenly perceive that the connection to the server has been lost.

#### Bypassing DRM

As noted above, DRM methods tend to have unique workarounds, depending on their specific characteristics. For non-interactive multimedia, one general approach is called the Analog Hole. The Analog Hole is a “flaw” in any DRM scheme, which is due to the fact that any media must eventually be consumed by a human. For example, a video will eventually have to be displayed on a screen and seen by someone’s eyes. In the simplest setting, a human could just record the protected music or movie with a microphone or a camcorder. Programs exist that will even record a movie as it is playing on the screen by scraping the screen’s pixels. However, since all the streaming media platforms known to us use lossy encoding for space and bandwidth-saving reasons, this type of DRM bypassing has the downside of a loss of quality due to the necessity to re-encode the captured audio and video. The only way to duplicate such content without quality loss is to capture the decrypted content after decryption but before decoding. There are two ways to do this: recovery of the keys used in the cryptographic process and the interception of the decrypted content. The former method requires approaches that may vary widely based on the DRM scheme and the type of encryption and key management used. Additionally, white box cryptography could be used to greatly complicate such an implementation by obscuring the usage of the cryptographic keys. The latter approach, which Movie Stealer uses, allows us to intercept decrypted content irrespective of the underlying encryption protocols. By doing this, it is possible to recover the original high-quality media sent by the media originator in a general way.

## **Chapter 2: Materials & Method**

## 2.1 – Planning of Experiment

There are various ways to embed data in videos. Hiding data into another data can be termed Steganography. There are many other famous ways of steganography like Least Significant Bit Steganography or LSB Steganography. Though the style of steganography here used is embedding a string that is made of only uppercase characters into a video, in a much simple way with the help of Python OpenCV.

Here the overall concept is:

- A Streaming service telecasts digital visual contents, but if they are pirated and telecasted in third party websites and peer to peer connections like telegram or torrent, it brings inevitable amount of loss for the streaming service.
- So, they use this new technology that not only hides data in their videos but also gives the hidden data a way to be recognized by the authority, to hide unique flags to each of their users.
- Next time they find a content own by them in a third-party server, they extract the unique string hidden in the video and backtrack the user who pirated it.

Now the Encryption Technology includes:

- Generating unique Flag strings for each unique user accounts.
- Using them to encrypt the video so that the flag string is hidden in the video – using Visual Morse Code Steganography.
- Getting a way to decrypt the video and find the hidden data, both by computers and by human resources.

We will be focusing on the Visual Morse Code Steganography and embedding the flag into the video in a certain position in form of visual morse code.

At first, the Morse Code and everything about it is described below.

## 2.1.0 – Details of Data

### Data used as the video

We managed to find a royalty free video that we can use to embed the piece of data we want to hide. So that will be the video file that we will use, the 'video.mp4' in the same directory. And the String that we are embedding in the video is manually Given by The User.

**YouTube link** - <https://youtu.be/IUN664s7N-c>

**And the Input Flag here used is:** DRM-Project-2021

Also, while taking the string input, we are converting every character of it into upper case characters, so that we can find the same values if morse code for the lower and the upper case

## 2.1.1 - Morse Code

It is basically a detailed system of dots, dashes, and spaces which is used to represent numbers, punctuations, and letters of the alphabet. It is generally used as a code but it can also be used to communicate without the ability or need to use actual characters. The codes are transmitted as electrical pulses of varied lengths or analogous mechanical or visual signals, such as flashing lights.

### **Morse Code has two versions**

The original version and the newer International Morse Code. The international system simplified things by removing the spaces and making all the dashes a standard length.

In short, Morse Code is used for representing letters of the alphabet, numerals, and punctuation marks by an arrangement of dots, dashes, and spaces.

### **The History of Morse Code**

Morse Code was invented by American Samuel F.B. Morse in the 1830's for use in telegraphy. The original version only represented numbers but his partner, Alfred Lewis Vail, improved it to include letters and some punctuation. Morse Code made its way over to telegraph operators in Europe when it was realized that changes were needed, as Morse Code was ill-equipped to deal with many of the diacritic characters in other languages. This led to the creation of International Morse Code in 1851, which has remained relatively the same till date.

### **Using Morse Code**

Main advantage of Morse Code that it can be used in any circumstance. It works as long as it is possible to create a signal of some sort, whether that be a written or symbolic image, flashing a light, or even just tapping on something. It is also used commonly in radio communication. Morse Code is considered an important part of training for soldiers due to its flexibility.

Obviously, language can still be a barrier, which is why a number of common words are considered key components of Morse Code. The most famous of these is "SOS," designed as a universal distress signal by the German government in 1905. The SOS Morse Code was actually just chosen for its simplicity, being represented by three dots, three dashes, and three more dots.

Morse Code was used extensively during World War II, in the Vietnam and Korean wars, and remained the standard format for ocean communication until it was replaced by the Global Maritime Distress Safety System in 1999.

### **Morse code chart**

Modern technology has reduced emphasis on learning and usage of Morse Code, but it still has the potential to be useful in severe or remote circumstances. It still remains popular among amateur radio operators and has proven an effective form of communication for those rendered incapable of speech by stroke or paralysis.

We are using a dictionary, where all the uppercase characters, symbols and numbers are saved as keys with the unique morse string dedicated for themselves as values. So, the encryption gets easy and with O (Length of String) time complexity.

### 2.1.1.1 – Morse Code Chart

|         | American<br>(Morse) | Continental<br>(Gerke) | International<br>(ITU) |
|---------|---------------------|------------------------|------------------------|
| A       | • —                 | • —                    | • —                    |
| Ä       |                     | • — • —                |                        |
| B       | — • • •             | — • • •                | — • • •                |
| C       | • • • •             | — • — •                | — • — •                |
| CH      |                     | — • — • —              |                        |
| D       | — • •               | — • •                  | — • •                  |
| E       | •                   | •                      | •                      |
| F       | • — • •             | • • — •                | • • — •                |
| G       | — • — •             | — • — •                | — • — •                |
| H       | • • • •             | • • • •                | • • • •                |
| I       | • •                 | • •                    | • •                    |
| J       | — • • — •           | — • • — •              | — • • — •              |
| K       | — • • —             | — • • —                | — • • —                |
| L       | — • — •             | — • — •                | — • — •                |
| M       | — • —               | — • —                  | — • —                  |
| N       | — •                 | — •                    | — •                    |
| O       | • •                 | • •                    | • •                    |
| Ö       |                     | — • — • •              |                        |
| P       | • • • • •           | • • • • •              | • • • • •              |
| Q       | • • — • •           | — • — • —              | — • — • —              |
| R       | • • • •             | — • — •                | — • — •                |
| S       | • • • •             | • • • •                | • • • •                |
| T       | — •                 | — •                    | — •                    |
| U       | • • •               | • • •                  | • • •                  |
| Ü       |                     | • • — • —              |                        |
| V       | • • • •             | • • • •                | • • • •                |
| W       | — • • —             | — • • —                | — • • —                |
| X       | • • — • •           | • • — • •              | • • — • •              |
| Y       | • • • • •           | — • — • •              | — • — • •              |
| Z       | • • • • •           | • • — • •              | • • — • •              |
| 1       | • — • • •           | • — • • •              | • — • • •              |
| 2       | • • — • •           | • • — • •              | • • — • •              |
| 3       | • • • — •           | • • • — •              | • • • — •              |
| 4       | • • • • —           | • • • • —              | • • • • —              |
| 5       | — • — • •           | — • — • •              | — • — • •              |
| 6       | • • • • •           | • • • • •              | • • • • •              |
| 7       | — • — • •           | — • — • •              | — • — • •              |
| 8       | • • • • •           | • • • • •              | • • • • •              |
| 9       | — • — • •           | — • — • •              | — • — • •              |
| 0       | — • — • •           | — • — • •              | — • — • •              |
| 0 (alt) | — •                 | — •                    | — •                    |

## 2.2 – Algorithm

We break the total way of Encryption into several parts.

- i. Morse Code Encryption and Decryption for a given string.
- ii. Processing a video Frame by Frame.
- iii. Processing a Frame Pixel by Pixel.
- iv. Manipulation of pixels in certain areas at certain frames so that the pixels can blink with time.

In this way we will get the video that is needed to be streamed to individuals.

## 2.2.1 - Logic details

Let us discuss the total algorithm with minute logic details in here. Here all the parts of the Algorithm are discussed. The code is broken into some different files existing in a same directory for making it easier to interpret.

### 2.2.1.1 - Morse.py

#### Morse Code Encryption and Decryption for a given string

##### Dictionary

Dictionary in Python is an ordered collection of data values, used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, Dictionary holds **key: value** pair. Key-value is provided in the dictionary to make it more optimized.

In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its **Key: value**. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be immutable.

##### Adding elements to a dictionary

In Python Dictionary, Addition of elements can be done in multiple ways. One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = 'Value'. Updating an existing value in a Dictionary can be done by using the built-in **update()** method. Nested key values can also be added to an existing Dictionary.

##### Dictionary methods

| Methods                         | Description   |
|---------------------------------|---|
| <u>copy()</u>                   | They copy() method returns a shallow copy of the dictionary.                      |
| <u>clear()</u>                  | The clear() method removes all items from the dictionary.                         |
| <u>pop()</u>                    | Removes and returns an element from a dictionary having the given key.            |
| <u>popitem()</u>                | Removes the arbitrary key-value pair from the dictionary and returns it as tuple. |
| <u>get()</u>                    | It is a conventional method to access a value for a key.                          |
| <u>dictionary_name.values()</u> | returns a list of all the values available in a given dictionary.                 |
| <u>str()</u>                    | Produces a printable string representation of a dictionary.                       |
| <u>update()</u>                 | Adds dictionary dict2's key-values pairs to dict                                  |
| <u>setdefault()</u>             | Set dict[key]=default if key is not already in dict                               |
| <u>keys()</u>                   | Returns list of dictionary dict's keys  |
| <u>items()</u>                  | Returns a list of dict's (key, value) tuple pairs                                 |
| <u>has_key()</u>                | Returns true if key in dictionary dict, false otherwise                           |
| <u>fromkeys()</u>               | Create a new dictionary with keys from seq and values set to value.               |
| <u>type()</u>                   | Returns the type of the passed variable.  |
| <u>cmp()</u>                    | Compares elements of both dict.   |



The code is given here -

```
MORSE_CODE_DICT = { 'A': '-.-.', 'B': '-...',
                    'C': '-.-.-', 'D': '-.-.', 'E': '.-.',
                    'F': '.-.-.', 'G': '--.', 'H': '....',
                    'I': '..-', 'J': '-.-.-', 'K': '-.-.-',
                    'L': '-.-.-', 'M': '--.', 'N': '-.-.',
                    'O': '---', 'P': '-.-.-', 'Q': '-.-.-',
                    'R': '-.-.', 'S': '...-', 'T': '-.-',
                    'U': '-.-.-', 'V': '...-.-', 'W': '-.-.-',
                    'X': '-.-.-', 'Y': '-.-.-', 'Z': '-.-.-',
                    '1': '.----', '2': '..---', '3': '...--',
                    '4': '....-', '5': '.....', '6': '-.....',
                    '7': '-.....', '8': '-.....', '9': '-.....',
                    '0': '-----', ' ': '-.-.-.-', ' ': '-.-.-.-',
                    '?': '-.-.-.-', '/': '-.-.-.-', '-': '-.-.-.-',
                    '(': '-.-.-.-', ')': '-.-.-.-'}

def encrypt(message):
    cipher = ''
    for letter in message:
        if letter != ' ':
            cipher += MORSE_CODE_DICT[letter] + ' '
        else:
            cipher += ' '
    return cipher

def decrypt(message):
    i=0
    message += ' '
    decipher = ''
    citext = ''
    for letter in message:
        if (letter != ' '):
            i = 0
            citext += letter
        else:
            i += 1
            if i == 2 :
                decipher += ' '
            else:
                decipher += list(MORSE_CODE_DICT.keys())[list(MORSE_CODE_DICT.values()).index(citext)]
                citext = ''

    return decipher
```

### Def encrypt (message)

In line 1 we can see the dictionary MORSE\_CODE\_DICT is initialized where the Morse code for the corresponding letter is written in the form of key: value format, Key being the letter and corresponding Morse code as its value. Next we have the definition of the function encrypt where we will encrypt our message in Morse code. In this function we take our message as argument. We take a blank string and initialize it as cipher. Now we check each letter in our message. For each letter in message we access the corresponding value from our dictionary i.e. for 'D': '-.-.', and insert it in our blank string 'cipher' followed by a space, if we encounter a space ' ' in the message we insert that space in cipher. And finally we return the string cipher which is our encrypted message.

**Def decrypt (message)**

In this function we will simply decrypt the encrypted message. This part of the code is for the receiver side. Here, we are going to use an approach called the index approach to decrypt the message as in a dictionary though it is easy to access the values using keys but one cannot access keys using values.

In the given code we 1<sup>st</sup> check if there is any space in our letter, if not we take the letter in our cipher txt string, if we encounter a space ' ', we increase the value of i by 1. Now if i is equal to 2, that means we have encountered 2 consecutive spaces and thus the end of encrypted message. Finally we will find the index of the values and then we form a list of keys and values of MORSE\_CODE\_DICT and then using that index as the reference, we return the corresponding value of the key from the new list formed to the string decipher i.e. for '-.-' index is 2, index 2 in list will be C. . We return the string as our decrypted message.

## 2.2.1.2 - Main.py

### 2.2.1.2 (a) - Processing a video Frame by Frame

#### OpenCV

OpenCV (Open-Source Computer Vision Library) is an open-source software library for computer vision and machine learning. OpenCV was created to provide a shared infrastructure for applications for computer vision and to speed up the use of machine perception in consumer products. OpenCV, as a BSD-licensed software, makes it simple for companies to use and change the code. There are some predefined packages and libraries that make our life simple and OpenCV is one of them

Gary Bradsky invented OpenCV in 1999 and soon the first release came in 2000. This library is based on optimized C / C++ and supports Java and Python along with C++ through an interface. The library has more than 2500 optimized algorithms, including an extensive collection of computer vision and machine learning algorithms, both classic and state-of-the-art. Using OpenCV it becomes easy to do complex tasks such as identify and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D object models, generate 3D point clouds from stereo cameras, stitch images together to generate an entire scene with a high resolution image and many more.

Python is a user-friendly language and easy to work with but this advantage comes with a cost of speed, as Python is slower to languages such as C or C++. So we extend Python with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. Doing this, the code is fast, as it is written in original C/C++ code (since it is the actual C++ code working in the background) and also, it is easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

Now this is the piece of code where we are processing a video frame by frame. In this, we have used OpenCV module and the **VideoCapture** function of the module.

- The Out variable is at first pointing to a NULL or None.
- The cap variable used is capturing or assigned in the video named 'video.mp4' from the same directory the main.py code is.
- The i variable means the number of frames while traversing through the video.
- The index variable is accessing the index of Emb array (That holds the Morse code stream in Binary format).

#### The Code is Given here -

```
out=None
cap= cv2.VideoCapture('video.mp4')
i=0
index=0
```

Then after that the while loop that is used is actually finding all the frames while the cap is opening. The ret and frame variables are assigned with cap.read(), which returns a variable pair. The first variable returns true if there is a next frame after the current frame, and returns false if the frame it is accessing in that very time is the last one, and the second variable sends the array format of the frames itself. So, if the ret variable is False, we get to know that the video has ended.

For ease of understanding we have made a directory named I, and have kept the images made from the frames using the serial number of the frame in the name of the image, so that we get to see all the frames as images in the folder.

And that one we are doing using `imwrite()` function, where the first parameter includes the name and the directory eventually of the respective image and the second parameter or argument is the frame itself from which the image is made.

Now we can get two options,

1. Making the second video which has morse code embedded into itself from the extracted frames or images.
2. Making the video just from the frames we are getting while scanning the frame.

In the 2<sup>nd</sup> option we don't need to save the extracted images. This helps us in some factors like,

- We don't need to sort the images.
- We don't need to use the module of `glob`, `re`.
- Space and Time complexity will be significantly reduced.

Still, we have taken the 1<sup>st</sup> approach and commented the part out from the whole code. In that we are changing the images and adding them again in a video after sorting them numerically. The process to make the second video is described afterwards.

After that we have to Embed the morse code in a time basis way so that we can see the morse code visually. So, that brings us in the next part, where we are using the function that we made: **`Embed(Frame, index)`**.

**The Code is Given here –**

```
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret == False:
        break
    cv2.imwrite('I/'+str(i)+'.jpg',frame)
    img = frame
    height, width, layers = img.shape
    size = (width,height)
    if out==None:
        out = cv2.VideoWriter('project.mp4',cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height))
    else:
        Embed(img,index)
        index+=1
        if index==len(Emb):
            index=0
        out.write(img)
        i+=1
        if i>500:
            break
out.release()
cap.release()
```

At last, we have to release the `cap` and `out` variables we have taken to point the video variables.

### 2.2.1.2 (b) - Processing a Frame Pixel by Pixel – Embed (Frame, index) Function

#### What is a pixel?

The word “pixel” means a picture element. Every photograph, in digital form, is made up of pixels. They are the smallest unit of information that makes up a picture. Usually round or square, they are typically arranged in a 2-dimensional grid.

#### RGB representation of a photo pixel

Each pixel consists of 3 RGB transistors which can store 8bits of information (0 to 255).The combination of 3 bit data forms one color of the pixel e.g. Teal (0,128,128), the computer will read it as R: 00000000, G: 10000000, B: 10000000 which will display teal color on that pixel.

In modern day we use the term FHD, 2k, 4k; these are nothing but the resolution of a screen. FHD means 1920 x 1080 pixels i.e. there are 1920 pixels along the length and 1080 along the width of the screen totaling to about 2million pixels. Here each pixel will have 3RGB transistors which will store 3sets of 8bit data (00000000 to 11111111) or (0 to 255) which will represent one color of the image we are trying to portray.

#### An image file is nothing but the RGB data for each pixel stored in a file.

OpenCV is the huge open-source library for computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today’s systems. By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human.

Point processing in spatial domain:

All the processing done on the pixel values. Point processing operations take the form –  
 $s = T(r)$

Here, T is referred to as a grey level transformation function or a point processing operation, s refers to the processed image pixel value and r refers to the original image pixel value.

#### Image Negative:

$s = (L-1) - r$ , where L= number of grey levels

#### Thresholding:

$s = L-1$  for  $r > \text{threshold}$

$s = 0$  for  $r < \text{threshold}$

Grey level slicing with background:

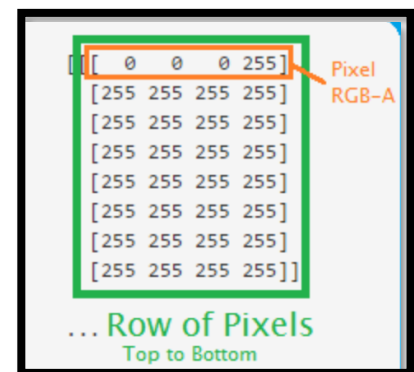
$s = L-1$  for  $a < r < b$ ,

here a and b define some specific range of grey level

$s = r$  otherwise.

For Describing the Embed Function, we have to know how the images are processed as a variable matrix of pixels.

Images are processed as 2 dimensional arrays of integers from 00 to 255, where in every pixel there are lists of 3 variables, the value of Red, the value of Green and the value of Blue.



**The piece of Code is -**

```
shade_factor = 0.2
tint_factor = 0.2
def Embed(img,index):
    if Emb[index]==1:
        '''
        img[10:20,10:30]=[255,255,255] #Just to whiten the pixels
        '''
        for a in range(10,15):
            for b in range(10,20):
                R=img[a,b,0]
                G=img[a,b,1]
                B=img[a,b,2]
                if R<=100 & G<=100 & B<=100:
                    newR = R + (255 - R) * tint_factor
                    newG = G + (255 - G) * tint_factor
                    newB = B + (255 - B) * tint_factor
                    img[a,b]=[newR,newG,newB]
                else:
                    newR = R*(1-shade_factor)
                    newG = G*(1-shade_factor)
                    newB = B*(1-shade_factor)
                    img[a,b]=[newR,newG,newB]
```

Here in the code, we are turning the pixels of the coordinate 10,15 to 11,20 (The total rectangle) to white for the parts where the Morse Code blinks. In the commented-out section you can see we had just made it 255,255,255 to all the RGB values, and the places becomes white. The only problem with that is, if the screen is automatically white, we will not understand if it is a blink or not.

Here in the effective code, we have designed a shade and tint logic with a factor of 0.2, and changed the shade of the given indices only. Now even if the blink is not visible in one time it will be visible after an interval of time again and again throughout the video.

**Manipulation of pixels in certain areas at certain frames so that the pixels can blink with time.**

### 2.2.1.2 (c) - Turning the Processed Data embedded frames to the Stream able video:

We discussed there are two ways to do this thing. Here are both the ways discussed in proper way.

- **This is how you make the videos from the extracted frames**

Here we need to use Glob module.

#### What is glob Python?

Even though the glob API is very simple, the module packs a lot of power. It is useful in any situation where your program needs to look for a list of files on the filesystem with names matching a pattern. If you need a list of filenames that all have a certain extension, prefix, or any common string in the middle, use glob instead of writing code to scan the directory contents yourself.

The pattern rules for glob are not regular expressions. Instead, they follow standard Unix path expansion rules.

There are only a few special characters: two different wild-cards, and character ranges are supported. The patterns rules are applied to segments of the filename (stopping at the path separator, /). Paths in the pattern can be relative or absolute. Shell variable names and tilde (~) are not expanded.

#### How to use Glob() function to find files recursively in Python?

Glob is a general term used to define techniques to match specified patterns according to rules related to Unix shell.

Linux and Unix systems and shells also support glob and also provide function glob() in system libraries.

In Python, the glob module is used to retrieve files/pathnames matching a specified pattern. The pattern rules of glob follow standard Unix path expansion rules. It is also predicted that according to benchmarks it is faster than other methods to match pathnames in directories. With glob, we can also use wildcards ('\*', '?', [ranges]) apart from exact string search to make path retrieval simpler and more convenient.

Again, the process to create the exact video is described below with the second process.

```
import glob
import re

out=None
numbers = re.compile(r'(\d+)')
def numericalSort(value):
    parts = numbers.split(value)
    parts[1::2] = map(int, parts[1::2])
    return parts

index=0
for filename in sorted(glob.glob('I/*.jpg'),key=numericalSort):
    img = cv2.imread(filename)
    height, width, layers = img.shape
    size = (width,height)
    if out==None:
        out = cv2.VideoWriter('project2.mp4',cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height))
    else:
        Embed(img,index)
```

```

        index+=1
        if index==len(Emb):
            index=0
        out.write(img)
    out.release()

```

- **This is the optimal way to do so while you are accessing the frames of the main video itself**

The Out variable pointing to None at first, when the framing was not started. So, when you start adding the data embedded video frames, at first you have to make a file where you mention the resolution, fps (In here we declared it as 25, depends on the number of frames and the duration of the given video), format (Mp4 here), and the name of the file itself. So, at first you assign a video pointer to out. After that the Out variable is not pointing at None anymore. So, from the second frame it is starting to embed values according the Embed function discussed earlier. After that, the out pointer just adding on the modified frames.

```

if out==None:
    out = cv2.VideoWriter('project.mp4',cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height))
else:
    Embed(img,index)
    index+=1
    if index==len(Emb):
        index=0
    out.write(img)
    i+=1
    if i>500:
        break

```

In this way, the video embedded with needed morse code will be there to stream to individual users with unique Flag strings.



## **Chapter 3: Result**

## 3.1 – Result

### Output At the terminal -

We printed out the morse code translation of the needed string so that we can check if the visual morse sinal embedded in the video is similar to that one or not.

The output screen after giving the input in the Main.py looks like –

```
PS C:\DRM Project - Infra> python -u "c:\DRM Project - Infra\Main.py"
Input for the Flag String that need to be encoded (All Caps) : DRM-PROJECT-2021

The Morse code representation of the Flag String -
-.. .-. -- -..... -.-. .-. --- -.-. -..... -.-. --- -.-.
```

### Output Video Clip -

After the code is completed running a video is created. Here in the report the video cannot be added, so we will attach a link where you can see the video for yourself. Here are some snaps where you can see the differences of the two videos.

Snap From video.mp4



Snap From project.mp4



Zoomed Snap From video.mp4



Zoomed Snap From project.mp4

Where the Blinking is happening and can be found in bare eyes



## **Chapter 4: Analysis & Conclusion**

## 4.1 - Analysis

We are successful to embed a meaningful morse signal in certain pixels of our video, and the blinks are either a tint or a shade version of the already existing colors in the pixel for a given time. And the data is embedded with more than 95% accuracy.

If anyone want to see the source code, 'video.mp4' which is the real video and the 'project.mp4', the data inserted video, may go to the **drive link**: (You have to have a mail id with @heritage.edu domain)

[https://drive.google.com/drive/u/1/folders/11m0Mp0\\_E-TR16lJOZZPefa2LPOp6LNwa](https://drive.google.com/drive/u/1/folders/11m0Mp0_E-TR16lJOZZPefa2LPOp6LNwa)

or the **GitHub link**:

<https://github.com/Infra17/Digital-Right-Management-System-with-Visual-Morse-Signal-Steganography>

or you can send an email to:

[rahit.saha.ece21@heritageit.edu.in](mailto:rahit.saha.ece21@heritageit.edu.in)

And the video Output you can get :

**Original Video YouTube link** - [youtu.be/IUN664s7N-c](https://youtu.be/IUN664s7N-c)

**Encoded Video YouTube link** - [youtu.be/yIUTjn0KIHY](https://youtu.be/yIUTjn0KIHY)

## **4.2 – Conclusion**

So, we have been successful to embed a data, and we can do some more research and developments so that the Algorithm becomes more complex, hard to decode or reverse engineer and more hidden in terms of bare eyes. But the more we make it not possible to understand, the more it is possible to be not found anyhow if the video gets compressed or degraded or edited in terms of pixels. That suggests, a long way still to go.

## 4.3 – Future Possible R&Ds

- **Changing or automating the pixels for the morse code:**

There is still a chance that the digital pirate will find out the spot and the technology and manipulate the part himself. We can have a dynamic positioning system depending on the total sum and positions of all the pixels, some basic machine learning model can help us in this part. Also, we can hide the video in the edge of the frames.

- **Changing the tint or shade level to get it hidden:**

For the project we have to make it easy to understand the difference in bare eyes, so we just made the factor 0.2, else to get it completely invisible we can just change some bits of the existing colors of the pixels. By that, we will be needing another script to search if we have any data embed there.

- **Scripting a Decoder Function:**

After our success of Encryption, we just need to write another code to decode if there is any morse code hidden in the video. The work is in progress. Future work will be visible in the GitHub link -

<https://github.com/Infra17/Digital-Right-Management-System-with-Visual-Morse-Signal-Steganography>

## References

Digital Rights management - <https://bitmovin.com/digital-rights-management-everything-to-know/>  
Digital Rights Management - [https://en.wikipedia.org/wiki/Digital\\_rights\\_management](https://en.wikipedia.org/wiki/Digital_rights_management)  
Morse Code - [https://en.m.wikipedia.org/wiki/Morse\\_code](https://en.m.wikipedia.org/wiki/Morse_code)  
Steganography - <https://www.ukessays.com/essays/computer-science/steganography-uses-methods-tools-3250.php>  
Steganography - <https://en.wikipedia.org/wiki/Steganography>  
How Netflix implements DRM - <https://medium.com/pallycon/how-netflix-protects-contents-part-1-a40508ed0001>  
OpenCV - <https://ieeexplore.ieee.org/document/6240859>  
OpenCV in python - <https://www.geeksforgeeks.org/opencv-python-tutorial/>  
Glob in python - <https://www.tutorialspoint.com/unix-style-pathname-pattern-expansion-in-python-glob>  
RGB representations - [https://en.m.wikipedia.org/wiki/RGB\\_color\\_model](https://en.m.wikipedia.org/wiki/RGB_color_model)  
Steganography - <https://www.youtube.com/watch?v=xepNoHgNj0w>  
Data Hiding in image - <https://www.youtube.com/watch?v=Ais5yNGSvHM>  
Guide to Video Steganography - [betterprogramming.pub/a-guide-to-video-steganography-using-python-4f010b32a5b7](https://betterprogramming.pub/a-guide-to-video-steganography-using-python-4f010b32a5b7)  
How Spotify implements DRM - <https://arxiv.org/pdf/2103.16360.pdf>  
Digital right management - <https://journals.sagepub.com/doi/10.1177/016555150202800504>  
Morse code - <https://internationaljournalofresearch.com/2020/07/15/history-of-morse-code-2/>  
Steganography - <https://www.worldscientific.com/page/journal-steganography>  
Extracting and Saving Video Frames using OpenCV-Python  
<https://theailearner.com/2018/10/15/extracting-and-saving-video-frames-using-opencv-python/>  
Image Pixel representation - <https://pythonprogramming.net/python-pixel-arrays/>  
Digital Right Management System - [https://www.widen.com/blog/digital-rights-management#:~:text=Digital%20rights%20management%20\(DRM\)%20is,prevent%20unauthorized%20modification%20or%20distribution.](https://www.widen.com/blog/digital-rights-management#:~:text=Digital%20rights%20management%20(DRM)%20is,prevent%20unauthorized%20modification%20or%20distribution.)

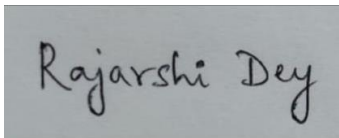
**Thank you.**

*Signature of the Students*

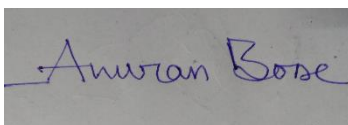
Rahit Saha



Rajarshi Dey



Anuran Bose



Saptarshi Roy Chowdhury

