FINAL YEAR PROJECT REPORT

FIT3036

Data Hiding in MKV Container Format

Maxim Zaika

(26437929)

Supervisor: Wong Kok Sheik

THE REPORT SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF COMPUTER SCIENCE

SCHOOL OF INFORMATION TECHNOLOGY

MONASH UNIVERSITY MALAYSIA

Data Hiding in MKV Container

October 19, 2017

Abstract

These days, video and audio content is multiplexed together, and get wrapped in a multimedia container format, such as MP4, MKV, FLV, etc. Due to increased usage of videos these days on various platforms, such as YouTube, Snapchat, Instagram etc., it gets a lot more challenging to verify the integrity of the video files (both video and audio). Build in authentication systems are no longer reliable, and can be easily manipulated by unauthorized parties. Meaning that it is a lot more challenging to prove the ownership of the file or even claim the copyright. This project proposes the idea of hiding the data in the form of letters or digits in in the multimedia container using the encryption algorithm. Since MKV (also known as Matroska) container is based on open project, and is able to output higher quality videos, store subtitles, and even store 3D videos, it has been decided to select the MKV container as the source of carrying the hidden data. The data will get hidden in both the video and audio stream of the MKV container, which will allow the rightful owner of the file to prove the integrity of the file and claim copyrights a lot easier. Multiple tests will be performed on various MKV format containers downloaded from the internet. Results suggest that storage of unlimited amount of characters, and insignificant file size increment were achieved.

Keywords:

mkv, data, embedding, matroska, encryption, parser

Acknowledgment

I, Maxim Zaika, would like to say thank you to my supervisor Dr. Wong Kok Sheik, who has always been there for me to assist me in such a challenging task. Without Dr. Wong’s help, I would never be able to understand the whole data hiding concept, and of course I would never be able achieve the goal of this project. Another person, who has tried his best to assist me in this challenging, but successful journey was Dr. Wong’s student April Pyone Maung Maung, who has successfully completed his master’s degree, where he, together with Dr. Wong, proposed the idea of data hiding in MP4 container. Finally, it would never be possible to get to this stage of my life, where I get a chance to develop something new by myself, without the help and motivational speeches from my beloved parents.

Table of Content

List of Figures

List of Tables

Chapter 1 Introduction

1.1. Background

The usage of the smartphones, computers and video streaming websites have been increased dramatically within few years. Due to increased number of video content, the usage will be rising for quite some time. There are 2.6 billion smartphone users globally, and this number is expected to be increased up to 6.1 billion by 2020 (Lunden, 2015). The fastest-growing video streaming website, called YouTube, achieves 100 million video views daily (Cheng, Dale & Liu, 2008). Since the recorded video file always includes the audio track, people tend to call the audio-visual content as a video. In order to achieve a single audio-visual file, the audio and video content are multiplexed in a container. Currently, there are many different container formats, for instance: MKV, MP4, FLV, WMV, MOV, WebM, etc., and each format has its own purpose. According to Orlin (2012), 69% of the web videos and 58% of the mobile videos use MP4 container; however, MKV (also called Matroska) aims to become the main multimedia container format because it is able to support higher quality audio, and it is based on open project, meaning that it is free for personal use (Gordon, 2012). Matroska container gradually progresses to become a universal envelope that is able to store the videos with the highest resolution (including 3D stereoscopic videos and subtitles), and as well as audio tracks with the highest bitrate (Smith, 2016). Even though Matroska container allows users to store full length movies or CD in a single file, it becomes a lot more problematic to verify the integrity of the video file (like MKV container). Some important or even private content needs to be stored securely from third parties, and the content's data needs to be protected from unauthorized modifications, and constantly has to be verified by the receiving party. To preserve the integrity of the private content like audio, video, image, or text from being modified, the technique of data embedding has been used as a potential solution in this project.

1.2. Objective

In order to accomplish the project, the main goal of understanding the structure of the MKV container has been set since the beginning. Understanding the structure of the container will allow me to identify the right place of where the data can be hidden. This means that the technique of the data detection needs to be implemented. By hiding the data inside the container users will be able to verify the integrity of the file; however, this process needs to be done securely, therefore, another goal of hiding the data securely has been set meaning that some sort of encryption algorithm needs to be used. The goal of finding out the right format of storing the data inside the container, such as hexadecimal or binary, has also been set in order to ensure the full functionality of the file. By combining all the goals above, the technique of data hiding needs to be developed. The final and the most important goal of executing the edited MKV container without any distortions has also been set, without accomplishing the final goal, the whole process would be meaningless.

1.3. Requirement and Constraint

1.3.1. Functional Requirements

The proposed system needs to be able to hide the desired information provided by the user based on Matroska’s structure. The process is performed by inserting the information, such as letters or digits, into the Matroska container.

1.3.2. Non-Functional Requirements

It is important to keep the information, such as the owner’s name, privately and securely away from the unauthorized access. While Matroska is able to store this kind of information, it is still unable to store it securely, therefore data hiding has been introduced.

1.3.3. Constraints

Since the MKV container is a relatively new type of container, it lacks information on both online and library sources, which makes it a lot more difficult to identify and figure out the right way of implementing the data embedding technique.

1.4. Organization of the Thesis

The first chapter of this report provides the background to the problem about all the containers that currently exist by providing some facts from reliable sources. It is followed by the objectives that need to be accomplished in this project, and provide the optimal solution, which is used later in the project. This chapter also provides diverse types of requirements, and as well as constraints faced during the research and development stages. Chapter 2 provides clear explanation of the MKV container, and its purpose, and also clearly explains the use of the proposed solution for this project. It is followed by detailed explanation of Matroska's Structure, which explains why the build-in sections of the container cannot be used as the potential solution. This chapter also provides the risks that have been faced during this project, the resource requirements, such as what machine that needs to be used to run the code and what programming language that has been used in this project, and finally shows the timeline of the completed project. Chapter 3 shows the diagrams of the internal design and software architecture, and as well explanation of some important pseudocodes. Chapter 4 contains the results that have been achieved, the disadvantages that have been faced, and as well as CPU and RAM usages, and estimated time complexities. Chapter 5 has some discussions that are considered to be important, and as well the recommendations on improving the system. Chapter 6 concludes the research, provides the contributions, and tell what future work can be done.

Chapter 2 Literature Review

2.1. Introduction

Back in 2008, the downloads of the Matroska container have exceeded 4 million. Matroska was developed during a project called MCF (Multimedia Container Format). Even though it derived from MCF, it is completely different because it uses a binary derivative of XML called EBML (Extensible Binary Meta Language). EBML ensures the extensibility of the format without damaging the file support of the previous parsers. Matroska is a container, which can also be called as an envelope, that allows users to store many audio, video, and subtitle streams in one file, for example, a complete movie or CD in a single file. MKV contains all the vital features that anyone would ever need in a modern container such as fast searching in the file, error recovery, and selectable subtitle, audio and video streams (Bornaghi & Damiani, 2008). Currently, Matroska container uses 4 different extensions, and each extension has its own advantage. The most common and the most known extension is MKV, which is used for video files (containing audio and video files only). Other extensions like MKA, MK3D, and MKS are less known and responsible for storing either audio files, stereoscopic 3D video, or subtitles only (Matroska FAQ, 2016).

As stated in Navas, Vidya, & Dass (2011), data embedding is the process where the data is inserted into a content without modifying the original content and allowing it to get executed as intended. The embedded data depends on the type of the application or the type of the content. Some personal data, such as owner's name or a watermark, is advised to be inserted. By looking at the Figure 2.1., the inserted watermark is not supposed to be easily detected by unauthorized party, while the authorized party must be able to identify the integrity of the file through locating the hidden watermark. Since data embedding is an effective method of preserving the integrity, it is also widely

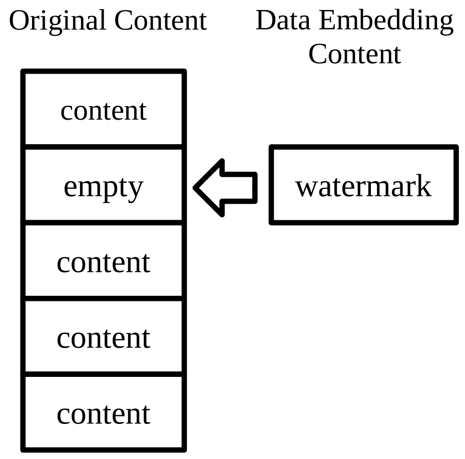


Figure 2.1.: Data Embedding Structure

used in securing the information in network protocols using the various methods (Tsai, Fan, & Chung, 2001). According to Tew, Wong, Phan, and Ngan (2016), to verify the ownership of the specific video or something that is happening in the video, the authentication and integrity processes need to be shown. While in this case, it is important to prove the integrity of the Matroska file, and all its contents (both video and audio). Therefore, implementation of the authentication scheme, such as data embedding, is crucial in preventing unauthorized parties gaining benefits or causing hatred to others. Since MKV container is able to store multiple contents, the best solution to prevent unauthorized parties from modifying the file, or in other words preserving the integrity, is to insert the authentication code into each media stream, such as video and audio streams, by the use of a unique key and knowing the location of the inserted code (Tsai, Fan & Chung, 2001; Maung, Tew, & Wong, 2016).

2.2. Structure of MKV Container

Matroska consists of a multi-level structure, which can be seen on the Figure 2.2. The purpose of the diagram is to show a simple structure, which explains how the MKV container works. More detailed version of the diagram can be found on Matroska's official website. The top level called Header, which contains the information that describes what EBML version has been used to create the MKV file. The next level Meta Seek Information is not required to be used; however, it is very useful for searching the entire file to find the elements like Tracks, Chapters, Tags, Attachments and others. Basically, Meta Seek is an index of all other levels. The Segment Information stores the basic data that is related to the whole file such as the title of the file or also called a unique ID that is used to identify the file. The Track level is straightforward and contains the information of each track (i.e., audio, video, subtitle) in the MKV file. The Chapters level stores the information of the chapters and allows to jump to a specific point of the video or audio file, while the Clusters level

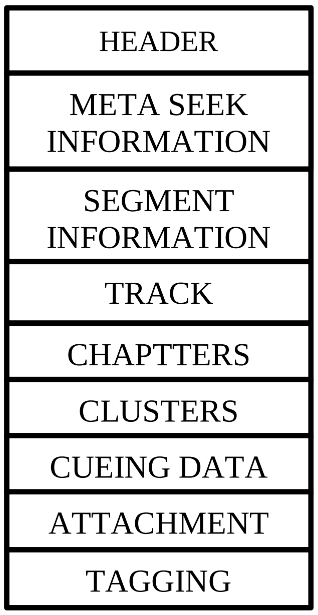


Figure 2.2.: MKV Structure

contains all the frames of all the videos and as well as audio of each track. This project proposes that Clusters level is the right place of storing the data, and the idea of storing the data will be described later. Cueing Data works as an index for each of the tracks that allows to seek through the track to a specific time. The Attachment level allows inserting various types of files such as videos, pictures, programs etc. Finally, the Tagging section contains the information such as the songwriter, actors of the video and other useful information (Diagram, 2016). While Tagging looks like the right section to hide the data, it is still insecure and can be changed almost by anyone. Therefore, the secured technique of data embedding needs to be used.

2.3. Project Risks

2.3.1. Insufficient Amount of Information

The first risk that has been met is insufficient amount of information or research about the Matroska container. The probability of this risk occurring has been expected to be high because MKV is still a relatively new container and there is not much information available about it. The risk reduction strategy that has been used is browsing through online libraries and precisely reading each available source of information, and mostly focusing on Matroska’s official website, which contains most of the required information and is also regularly updated.

2.3.2. Difficulty in funding functional and suitable MKV Parser

The second risk that has been faced is difficulty in finding fully functional, and suitable MKV parser in Python Language for this kind of project. The probability of this risk happening has been expected to be medium due to insufficient knowledge in EBML programming language, because all the Matroska containers have been originally written in EBML language, and then converted to any other language but Python because Python, just like MKV container, is a new type of language that is not used as frequently as other languages. At first, the idea of writing absolutely new MKV parser from scratch has been taken into a consideration; however, this would take a lot of time and the goal of the project would never been achieved. Therefore, the risk reduction strategy that has been used is browsing through online websites in hope of finding a functional and suitable MKV parser, which would allow this project to begin the implementation of data embedding.

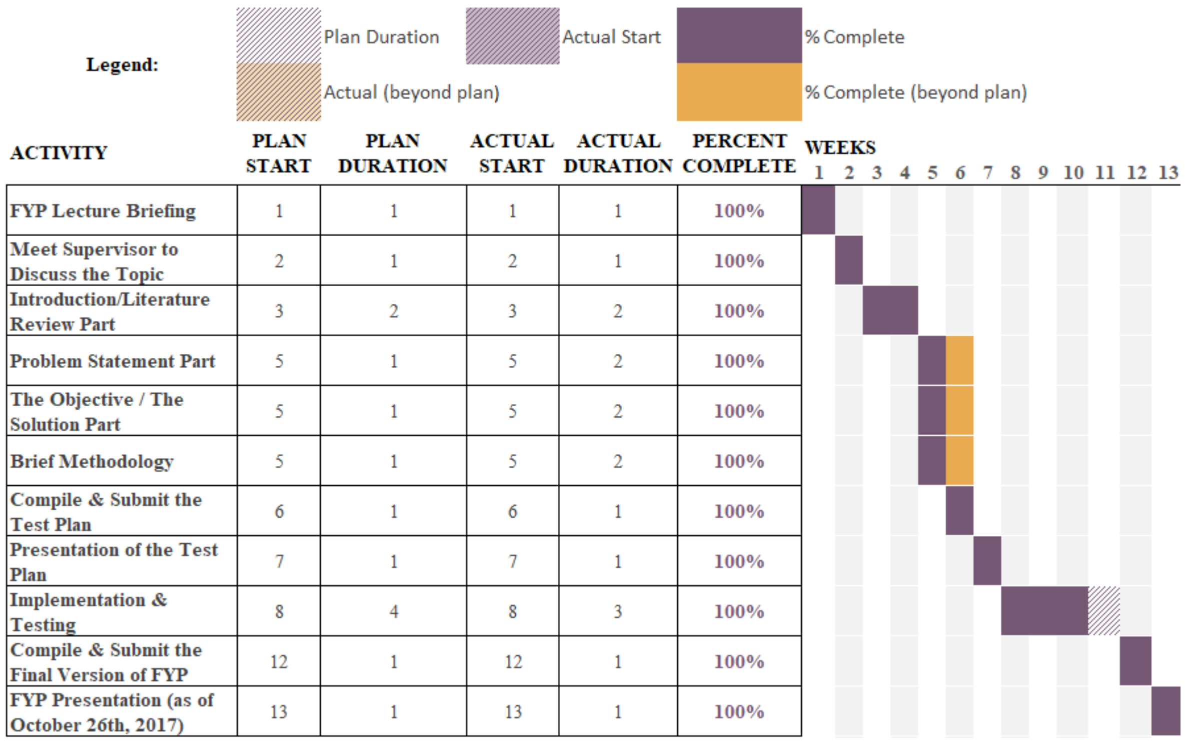
2.3.3. Incompatibility of Operating System

The final and the most significant risk that has been faced is switching the operating system of the project. Originally, there was no any probability of this risk happening; however, it would be logical to say that the probability of this risk occurring had to be high because most of the programming is performed on Unix machines. Therefore, during the development stage, it has been decided to accept the risk, install the Unix machine on PC, and learn the Unix machine from scratch, due to not having knowledge in it previously.

2.4. Resource Requirements

Out of all the languages, Python (ver. 3.6.2) has been used to code the program, which allows users to hide the desired data inside the MKV container. Originally, it has been decided to use a Window 10 machine, and it has also been expected that the project would be able to be executed on all of the machines; however, during the development stage, the idea of using a Windows 10 machine has been altered to using a Unix machine. In this case, Ubuntu has been used as a Unix machine, but Macintosh is also able execute the code after installing the Python. This has been decided because the MKV parser can only be executed on Unix machines, therefore, the data embedding code has also been developed to be executed on Unix machines. Even though the data embedding code can be modified to be executed on Windows 10 machine, the Parser still needs to be executed on a Unix machine.

2.5. Timeline



Legend: ▨ Plan Duration ▨ Actual Start 🟪 % Complete ▨ Actual (beyond plan) 🟧 % Complete (beyond plan)

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE	1	2	3	4	5	6	7	8	9	10	11	12	13
FYP Lecture Briefing	1	1	1	1	100%	🟪												
Meet Supervisor to Discuss the Topic	2	1	2	1	100%		🟪											
Introduction/Literature Review Part	3	2	3	2	100%			🟪	🟪									
Problem Statement Part	5	1	5	2	100%					🟪	🟧							
The Objective / The Solution Part	5	1	5	2	100%					🟪	🟧							
Brief Methodology	5	1	5	2	100%					🟪	🟧							
Compile & Submit the Test Plan	6	1	6	1	100%						🟪							
Presentation of the Test Plan	7	1	7	1	100%							🟪						
Implementation & Testing	8	4	8	3	100%								🟪	🟪	🟪	▨		
Compile & Submit the Final Version of FYP	12	1	12	1	100%												🟪	
FYP Presentation (as of October 26th, 2017)	13	1	13	1	100%													🟪

Figure 2.5.: Gantt Chart

Figure 2.5. illustrates the timeline of the completed project by the use of the Gantt chart. On the right side of the Gantt Chart, the graph displays the durations of all the activities. The “Plan Start” and “Actual Start” sections display the week number of when the specific activity begins, while the “Plan Duration” and “Actual Duration” display the number of weeks it will take or has taken to accomplish the specific activity. By developing the Gantt Chart for this project, it has been assumed that the final presentation has already taken place, therefore, the percent complete has been set as 100%.

Chapter 3 Method

3.1. Introduction

The whole MKV container is divided into block timecodes represented in 16bit signed integers (known as sint16). By using signed integers, the range of the block timecodes can be set between -32768 and +32767 units. The block timecode represents the raw timecode related to the cluster's timecode. In chapter 2, it has been explained that the level Cluster contains all the frames, or in other words the multimedia data of the frame, and it usually has a duration of a few seconds. The raw timecode gets multiplied by the timecode scale, which converts the block timecodes into nanoseconds. There is a possibility of having a negative raw timecode, whenever the cluster's timecode is set to zero; however, this timecode gets ignored, but is still stored within the MKV container. Therefore, this project proposes the storage of hidden data inside the clusters by allowing the user to select the specific cluster's timecode, such as -3.05, 2.10, 3.315. The range of the cluster's timecodes depends on the length of the video file inside the MKV container.

3.2. Internal Design

Since the data embedding mechanism is proposed in this project, it is also important to have a mechanism that is able to verify the integrity of the modified MKV container by allowing the user to detect the hidden data. Therefore, in order the ensure the full functionality of the proposed system, it has been decided to divide the system into two different systems with various instructions. First system allows user to hide the data inside the Matroska container, while the second system allows the user to reverse the process and verify the integrity of the file by inputting all the information that has been entered during the data hiding process.

3.2.1. Data Embedding Design

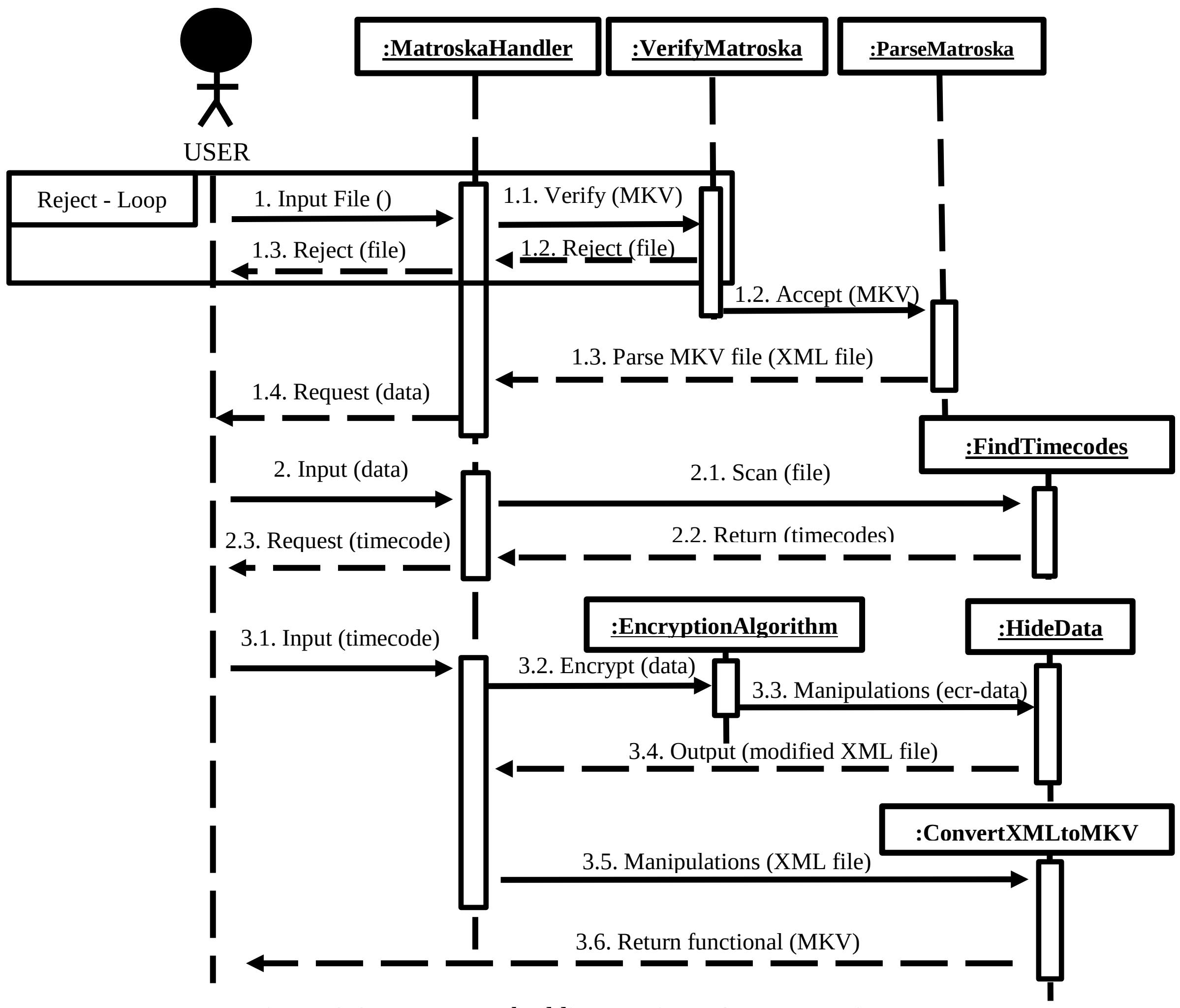


Figure 3.2.1: Data Embedding Design - Sequence Diagram

As described previously in Chapter 1, the **Tagging** level is important for storing some other information such as the owner of the file, or the actors of the video; however, this section can be easily modified by unauthorized party which makes it unreliable. Therefore, the method of data embedding, or also called data hiding is introduced in this project.

Figure 3.2.1. illustrates the sequence diagram, which explains the functionality of the Data Embedding system. The system needs to be coded the way that it is able to insert any watermark, such as owner's name, into the Matroska container without anyone knowing its location (known as timecode) except the owner. The system needs to request for any MKV file, which is selected by the user, and then it needs to verify whether the file is actually a Matroska container. Once the previous process has been performed and accepted successfully, the system needs to parse through the MKV file and return the XML version of the MKV container, requesting user's data and the encryption key. Right after the user's input has been successful, system identifies all the timecode regions and requests user to input the desired timecode (the number of timecodes depends on the length of the video file). Once user inputs the desired timecode, system needs to be able to write the data in, and output the functional MKV file.

3.2.2. Data Detecting Design

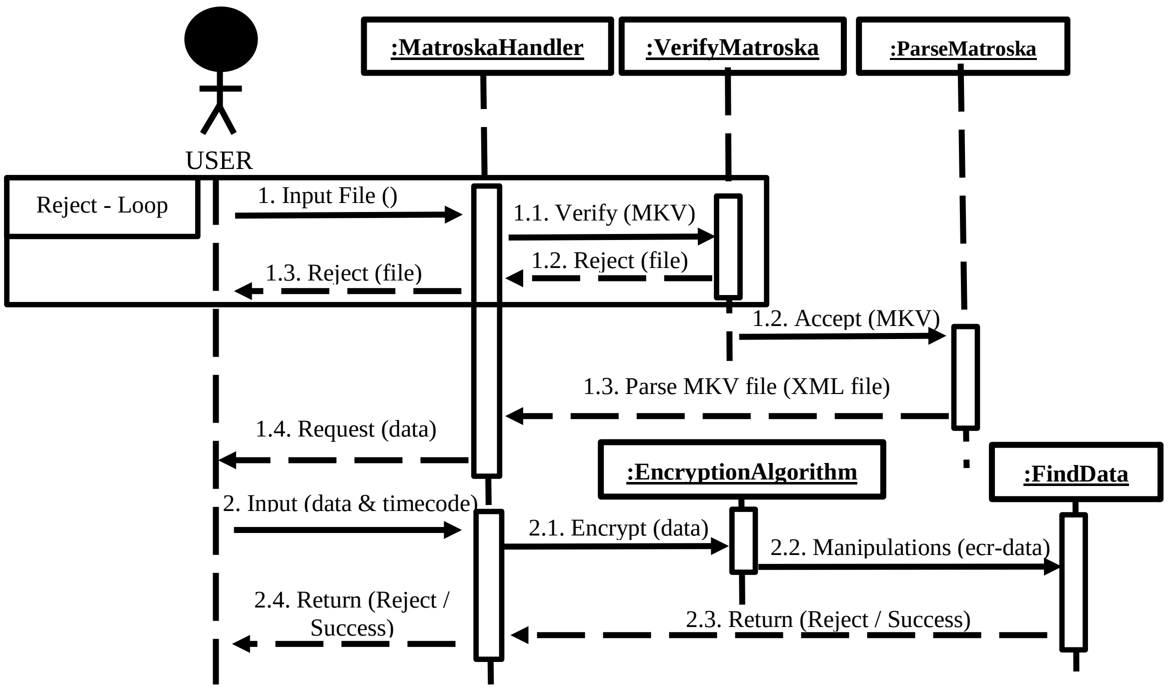


Figure 3.2.2.: Data Detecting Design - Sequence Diagram

Figure 3.2.2. illustrates the sequence diagram, which explains the functionality of the Data Detecting system. The system needs to be coded the way that it is able to detect any watermark, such as owner's name, by asking the user to input the watermark, the key, and the location (known as a timecode) that the user has entered during the data hiding process. First of all, just like during the data hiding process, system needs to accept and verify the MKV file. After that system needs to parse the MKV container and convert it to the hidden XML file, and after that ask the user to input all the required data that has been listed out earlier. All the data (except the timecode) gets encrypted and then the system starts searching for the encrypted text with the key that user has entered inside the selected timecode. If the key is wrong, or some of the text does not match, or the timecode is wrong, the system needs to reject user's inputs without letting the user know the exact issue that has been faced, such as a wrong key, text, or a timecode. By simply rejecting the user's input, the 3rd party users will not be able to identify where the issue is coming from, and will not be able to find the backdoor. However, if the all the data matches, then system needs to let user know that data has been detected successfully.

3.3. Software Architecture

3.3.1. Software Architecture of Data Embedding System

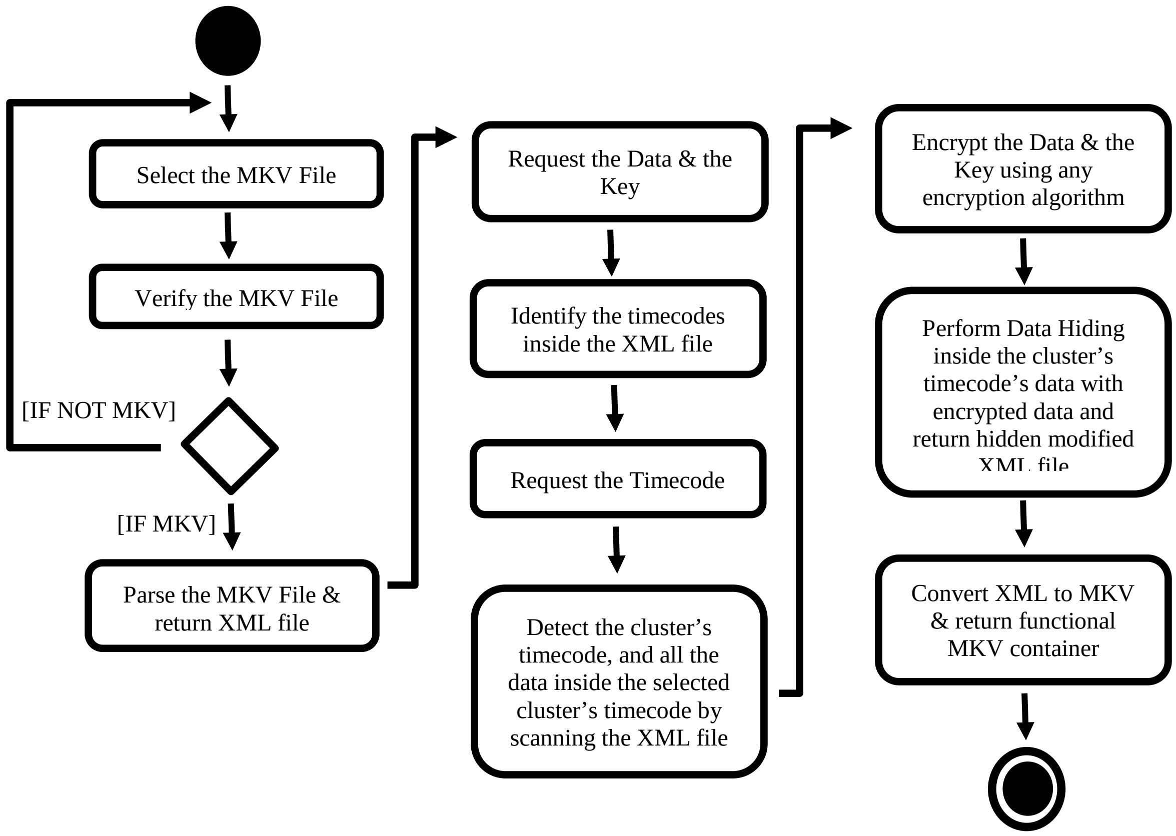

Select the MKV File
Verify the MKV File
[IF NOT MKV]
[IF MKV]
Parse the MKV File & return XML file
Request the Data & the Key
Identify the timecodes inside the XML file
Request the Timecode
Detect the cluster's timecode, and all the data inside the selected cluster's timecode by scanning the XML file
Encrypt the Data & the Key using any encryption algorithm
Perform Data Hiding inside the cluster's timecode's data with encrypted data and return hidden modified XML file
Convert XML to MKV & return functional MKV container

Figure 3.3.1.: Software Architecture of Data Embedding System - Activity Diagram

As it has been described earlier, the cluster's timecode is the right place where the data needs to be stored because during the parsing process, the MKV container gets broken down into multiple cluster timecodes depending on the length of the container's video. Once the small block of timecode's data (also called frame) gets detected, the lacing technique gets utilized. By using this technique, the space of the container can be saved and the data can be spread throughout the whole container in the most efficient way possible (Noe, 2009). Currently, there are 3 types of lacing: Xiph, EBML, which is inspired by EBML language, and the fixed-size lacing (Specifications 2017). By combining everything above, the parser creates hidden from the user XML file, which contains all the required and easily editable MKV container's information. The file can also be stored in TXT format; however, it has been decided to use XML because it can be easily executed using any browser, and it also allows to hide the unnecessary tags inside the file without removing them. After that, the data hiding algorithm with all the available timecodes get executed, where it requests the user to input the text that needs to be hidden, and the encryption key to ensure the safety of the hidden data. During this process, the data gets encrypted and converted to hexadecimal format, because the cluster's timecode data is stored in hexadecimal format, and the gets written into the XML file. As can be seen on Figure 3.3.1., the XML file gets converted back to MKV container by reversing the parser's process. To ensure that no hidden files have been left behind, the algorithm automatically removes all the hidden files.

3.3.2. Software Architecture of Data Detecting System

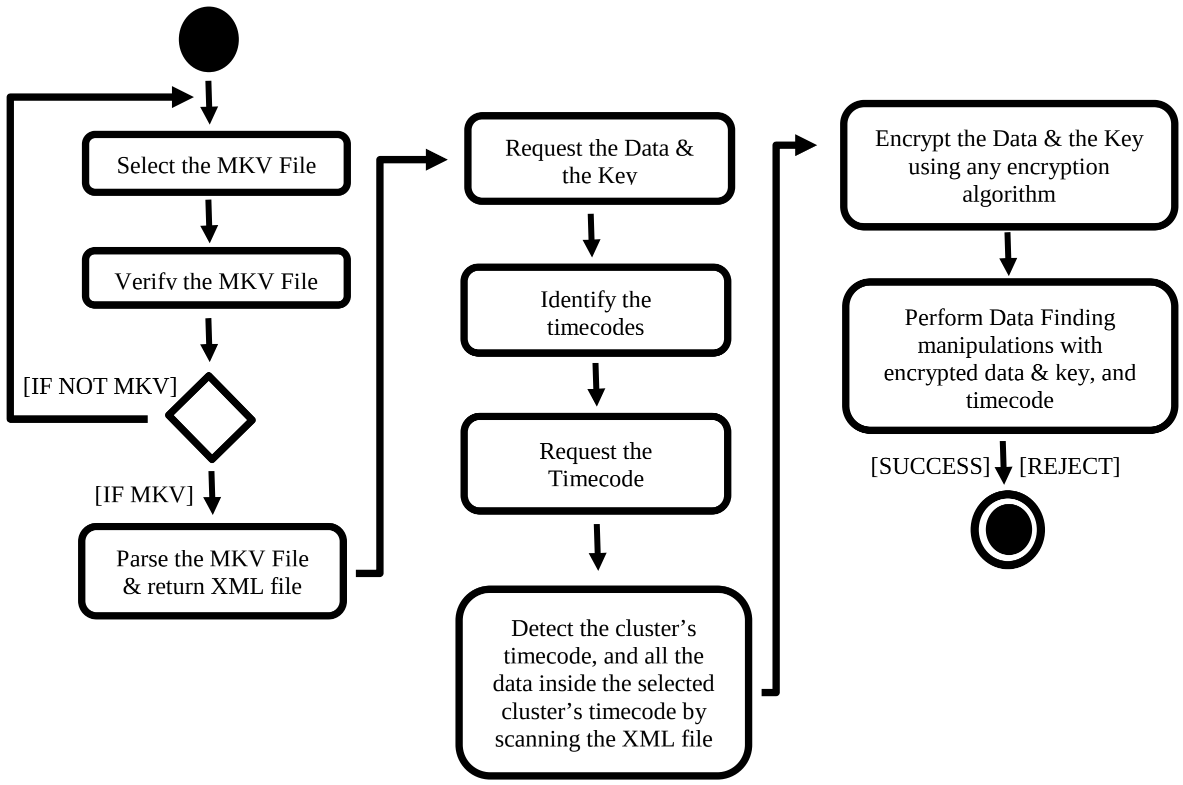

Select the MKV File
Verify the MKV File
[IF NOT MKV]
[IF MKV]
Parse the MKV File & return XML file
Request the Data & the Key
Identify the timecodes
Request the Timecode
Detect the cluster's timecode, and all the data inside the selected cluster's timecode by scanning the XML file
Encrypt the Data & the Key using any encryption algorithm
Perform Data Finding manipulations with encrypted data & key, and timecode
[SUCCESS]
[REJECT]

Figure 3.3.2.: Software Architecture of Data Detecting System - Activity Diagram

Figure 3.3.2., illustrates almost identical procedures as the data hiding process with some slight changes. The data detecting algorithm requests user to input the MKV container, after which algorithm automatically converts the Matroska container to the hidden XML file, and asks the user to input the text, the encryption key, and the timecode that have been used during the data hiding process. Once all the data has been successfully entered by the user, the data gets encrypted once again, and the algorithm accesses the selected timecode to verify the integrity of the entered data. It is important to input the identical information to ensure the successful verification. If the data cannot be detected inside the cluster's timecode, the algorithm returns the errors and removes the hidden XML file, and if the data has been detected, the algorithm returns 'success' and removed the hidden XML file.

3.4. Block Layout and Lacing Pseudocode

```
Function Block {
        Set vint → Track number
        Set sint16 → Timecode  // relates to a Cluster timecode
        Set int8 → Flags          // can be lacing, keyframe, discardable (something unnecessary)
        If (lacing is True) {
                Set int8 → frame count – 1
                If (lacing is EBML Lacing) {
                        Set vint → size[0]
                        Set svint → size [1..frame_count – 2]
                } else if (lacing is Xiph Lacing) {
                        Set int8 size[(<leading (frame count – 1) frames > / 255) + 1]
                }
        }
        Set int8[] → data
}
```

The pseudocode above is the short representation of the MKV parser, which converts the file to the XML file, and allows to perform further modifications. Development of the parser was not the main goal of this project; therefore, the code and pseudocode have been borrowed from the online source, and have been written by Shukela, 2012 and Noew, 2009 based on Matroska's official website.

3.4.1. Xiph Lacing

Inside the xiph lacing, the size of each frame (known as timecode) is coded based on a sum of int8. All the frames are broken down into small bytes. If the value before 255 bytes inside the size table is smaller, then it indicates the beginning of the new frame, while all the values before 255 get combined into an individual frame. The figure 3.4.1. illustrates the break down structure of the xiph lacing. For example, if the size of all the frames is broken down into {145, 200, 255, 3, 255}, then there are 3 frames with 345, 258, and 255 bytes.

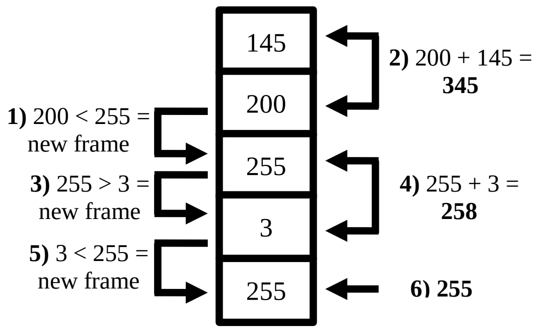


Figure 3.4.1.: Xiph Lacing Example

3.4.2. EBML Lacing

The EBML Lacing has borrowed the idea from the EBML language. The first frame of a lace, which is 0, equals size[0], while all other subsequent frames *i* of a lace are equal size[*i*] – size[*i*-1].

3.4.3. Fixed-Size Lacing

Whenever all the frames in a lace are of the same size, the fixed-size lacing technique is used. Since the size is fixed the calculation of the size of one frame is straightforward.

3.5. Data Embedding Pseudocode

```
Function main {
    Set a_word → "word that needs to be written in"
    Set encryption_key → "key used for the encryption"
    If (length of encryption_key) < (length of a_word) {
        Set encryption_key → same length as a_word

    }
    Set a_word → convert to ascii
    Set encryption_key → convert to ascii
    Set ascii_a_word → convert to hexadecimal
    Set ascii_encryption_key → convert to hexadecimal
    If (length of ascii_encryption_key) < (length of ascii_a_word) {
        Set ascii_encryption_key = → same length as ascii_a_word
    }
    Encrypt a_word(ascii_a_word, ascii_encryption_key, a_word) // encrypt ascii word using
                                                               // original word
    Convert encrypted word to hexadecimal // because MKV parser accepts hexadecimals
                                          // inside the cluster's timecode
    Timecodes = [] // read the XML file and identify the timecodes
    Set input_timecode → "input timecode here"
    Call function embed data (filename, input_timecode, encrypted_word_in_hexadecimal_format)

}

Function embed data {
        Loop through the file {
                Identify the location of the timecode {
                        Identify the location of the data inside the cluster's timecode {
                                Write-in the data
                        }
                } else not found timecode {
                        Try again
                }
        }
}
```

Above pseudocode is the incomplete representation of the data embedding algorithm that explains how the proposed algorithm works, and shows the contribution in the field of data hiding in MKV container.

3.6. Data Detecting Pseudocode

```
Function detect data {
    Set hexadecimal_word → ‘the encrypted word’ \\ basically the identical process like in data
                                                \\ hiding process
    Loop through the file {
        Loop each line of the file {
            Identify the location of the timecode {
                Identify the data inside the cluster’s timecode {
                    Read through the line ignoring first 6 characters // format
                }
                If there is at least 1 miss-match {
                    Return error
                } else fully matched {
                    Return success
                }
            }
        }
    }
}
```

Above pseudocode is the incomplete representation of the data detecting algorithm.

3.7. Encryption Algorithm Pseudocode

```
Function encryption algorithm {  // assume that we are converting the ascii_word using ascii_key
    Set encryption → ‘’
    for i loop in range of the length of the ascii_word {
        Set key_value → (i) mod (length of ascii_word)
        if the end of the ascii_key has been reached reset {
            Set key_value → size of ascii_word – key_value
        }
        Set key_mod → ascii_key[key_value]
        Set encryption → ascii_word[i] + key_mod + 33 // 33 because ascii values before 33
                                                      // are irrelevant
    }
    return encryption
}
```

The data embedding algorithm has been designed the way so that it allows users to replace the encryption algorithm to any algorithm they want, and since the strong encryption algorithm has not been set as the main goal of the project, it has been decided to use a newly created encryption algorithm from scratch. The above algorithm does not have any collision checking, meaning that algorithm is not reliable and can be easily manipulated, therefore, it is suggested to replace the above algorithm with any other desired algorithm. The algorithm has been simply used as a test version of the whole data embedding code to ensure that the use of encryption algorithm is possible.

3.7.1. Encryption Algorithm Explained

Let *X* be the plaintext, *K* be the encryption key, and *T* be the length of *X*. Firstly, outside of the encryption algorithm, the *K* concatenates with itself as many times as necessary in order to make it at least as long as *X*. Before encryption begins, both X and K get converted to ascii to ensure the compatibility with data embedding algorithm. The result is a string whose *i*-th letter is

$$X_i + K_{(i \bmod T)} + 33$$

Once the string is encrypted, it gets converted to hexadecimal to ensure the compatibility with data embedding algorithm once again. For example, if the plaintext is

‘hello’ or *‘104 101 108 108 111’* in ascii,

and the key is

‘maxim’ or *‘109 97 120 105 109’* in ascii,

then the ciphertext will originally be displayed in ascii as

‘35 33 46 43 40 35 34 42 50 41 34 41 43 43 35’

and then gets converted to hexadecimal as

*$!.+(#”*2)”)++#*

Chapter 4 Results

4.1. Introduction

During the 12 weeks development and researching process, the goal of executing the MKV container, without distortions, has always been the main priority; however, it would never be possible to achieve this goal without completing all the other goals. The overall results of the project are successful and show that there is a possibility of hiding the data, such as letters or digits, inside the MKV container in order to be able to verify the integrity of the file. A lot of testing has been performed on various MKV containers taken from the internet. In order to ensure that there are no copyright issues are faced with the videos, the safest and reliable online source called Sample Videos have been used (Download Sample Videos, 2017). The website provides the samples of different files with different file sizes. Therefore, multiple MKV container samples have been downloaded to perform data hiding and data detecting manipulations.

4.2. Results Obtained

4.2.1. Limited Number of Cluster's Timecodes

After converting the MKV container to the XML file, and trying to modify different cluster's timecodes, it has been concluded that modifying more than two cluster's timecodes cause slight video distortion; however, modifying even more timecodes causes both video and audio distortions. Since the main goal of hiding the data has been achieved, and to ensure that MKV container is executable without absolutely any distortions, it has been decided to reduce the selection of timecodes to one only.

4.2.2. Embedding Capacity

Since the cluster's timecode is the right place to store the data, the testing on the number of characters that can be hidden in the MKV container has also been performed. After trying to input up to 2,500 characters, there was no sign of any distortions detected in the MKV container at all. It is assumed that 2,500 characters is more than enough for the user; however, it is not suggested to use substantial number of characters because it reduces the safety of the encrypted text.

4.2.3. File Size Increment

Currently, the only disadvantage right now is the increased file size; however, it does not increase significantly, and does not affect the video playback. For example, if the original file size of the MKV container is 2.1 MB (2,097,641 bytes), then after modifying the file, and hiding 2,500 characters, the file size of the MKV container increases to 2.1 MB (2,122,058 bytes). From the values above, it can be calculated that the size has increased by 23,417 bytes or by 1.00%, which is considered not a significant difference, and does not carry any negative effects.

4.3. Performance

4.3.1. Data Embedding Algorithm

4.3.1.1. Performance Characteristics of Data Embedding Algorithm

Using the build-in python module, the CPU and the memory (also known as RAM) usage can be calculated. After hiding 2,500 characters of data inside the MKV container with the size of 21 MB (20,979,826 bytes), it has been calculated that data Embedding algorithm consumes 1.2% out of 100% of CPU performance, and as well 23.0% out of 100% of RAM. Testing on files with larger size have been performed, and it has been detected that CPU and RAM usage does not get increased. Therefore, it does not have negative effects on the system and safe to use.

4.3.1.2. Estimated Time and Space Complexities of Data Embedding Algorithm

Table 4.3.1.2.: Estimated Time Complexity of Data Embedding Algorithm

Function Name	Best / Worst Time Complexity	Overall estimated
pumpKey	Best & Worst = O (n)	Best & Worst = O (n^2)
text2ascii	Best & Worst = O (n)	
convert2hex	Best & Worst = O (n^2)	
encr_ascii	Best & Worst = O (n)	
get_timecodes	Best = O (n) & Worst = O (n^2)	
return_timecodes	Best = O (n) & Worst = O (n^2)	
hide_data	Best = O (n) & Worst = O (n^2)	

The estimated space complexity in this case is estimated to be O(n) due to storing the data in different arrays.

4.3.2. Data Detecting Algorithm

4.3.2.1. Performance Characteristics of Data Detecting Algorithm

Using the build-in python module, the CPU and the memory (also known as RAM) usage can be easily calculated. In order to ensure that the process is fair, the data hiding process has been reversed, meaning that exactly the same amount of characters have been entered (2,500), and the previously modified MKV container with the size of 21 MB (20,993,372 bytes) has been selected. Therefore, with all the data entered, it has been calculated that data embedding algorithm consumes 1.0% out of 100% of CPU performance, and as well as 23.1% out of 100% of RAM. Testing on files with larger size have been performed, and it has been detected that CPU and RAM usage does not get increased. Therefore, it does not have negative effects on the system and safe to use.

4.3.2.2. Estimated Time and Space Complexities of Data Detecting Algorithm

Table 4.3.2.2.: Estimated Time Complexity of Data Detecting Algorithm

Function Name	**Best / Worst Time Complexity**	**Overall estimated**
pumpKey	Best & Worst = O (n)	Best & Worst = O (n^2)
text2ascii	Best & Worst = O (n)	
convert2hex	Best & Worst = O (n^2)	
encr_ascii	Best & Worst = O (n)	
find_data	Best & Worst = O (n^2)	

The estimated space complexity in this case is estimated to be O(n) due to storing the data in different arrays.

Chapter 5 Discussion and Recommendations

5.1. Discussion

5.1.1. Motivation Behind Such Study

Out of all the proposed studies, I have decided to take the study where coding plays a significant role, and having some knowledge in using the MKV container previously, I have decided to take such study. I have always thought that all the containers, including MKV, have a single goal of storing the video and audio files and letting the video players, such as Windows Media Player, to run the files that are stored inside the container; however, I was wrong. By using the containers, people are able to transport some important information between each other, and can even be used for illegal manipulations, but that is not the intension of this project. The idea of developing some sort of algorithm, where the copyright claims can be easily performed, was always in my mind. The study of MKV container looked like the right place of where such algorithm could be developed.

5.1.2. Existing Studies

Before selecting the study, the research has been performed in order to gain some basic knowledge about the topic; however, there was no information found in both the library online library. During the development stages, another topic about the MP4 container has been found, which proves the possibility of hiding the data inside the MP4 container; however, MKV container is completely different from MP4, therefore, it requires some extra work and work from scratch. Even though there are no other data embedding in MKV container studies, the whole idea can still be supported with the data hiding in MP4 container (Maung, Tew & Wong, 2016). The only useful and the most practical source of information was the official Matroska website, which is updated on daily basis (Specifications, 2017).

5.1.3. Inability to Implement the Improved Version of the Algorithms

Current data detecting algorithm requires the user to input the text, the encryption key, and the timecode, which makes it unreliable because the only person who will be able to prove the existence of the hidden data is the rightful owner. Other users are also able to enter the same data and verify the integrity; however, this way other users will know the actual text that has been entered earlier. I have attempted to make it the way so that encryption key and the timecode are required only to identify the existence of the text. In order to do that, I had to modify the data embedding algorithm and as well as the data detecting algorithm, but I was unable to present the improved version, therefore this idea has been added to the recommendation list.

5.2. Recommendations

5.2.1. Modify or Replace the Encryption Algorithm

Currently; the encryption algorithm does not have any collision checking which makes it weak and unreliable, therefore, it is suggested to replace the encryption algorithm in order to increase the safety of the hidden data.

5.2.2. Improve Data Hiding and Detecting Algorithms

In data detecting algorithm, user needs to provide the string that has been entered during the data hiding process, meaning that in order to identify the data, user will have to enter the exact text in order to identify the data, which is unreliable, and can cause issues if user forgets the exact text. Therefore, it is suggested to make changes in the data hiding and detecting algorithms so that user does not need to enter the text, but the encryption key and the timecodes only, which will automatically detect any fully encrypted text and maybe even decrypt it.

5.2.3. Increase the Number of Cluster’s Timecodes

In this project, users are able to select one cluster’s timecode in order to hide their encrypted information, which is already a success; however, more tests need to be performed with different file sizes and number of characters in order to let the user to select more cluster’s timecodes, and hide even more data than originally suggested.

Chapter 6 Conclusion

6.1. Overview

In this project, the data hiding algorithm for MKV container has been proposed by utilizing the build in cluster's timecodes, and as well as the functional MKV parser that is able to convert the MKV container to an XML file. Since there are a lot of timecodes, users are able to select one desired timecode to hide and even detect the data using the developed data embedding and detecting algorithms. Now, users are easily able to verify the integrity of their files (both audio and video) in the MKV container, and prove that the container belongs to them, meaning that unauthorized users will not be able to steal someone's content. Therefore, rightful owners are able to claim the copyright without any difficulties. Experiment results verified that the proposed system can take the MKV container of any size as a whole to embed the data of unlimited capacity (suggested < 2,500 characters); however, the proposed system costs negligible file size increment (i.e. <= 1.0% for the whole MKV container), and allows to embed the data into one timecode only.

6.2. Achievements and Contributions

The research, using the functional MKV parsers, showing that the data needs to be hidden in any of the cluster's timecodes in MKV container has been contributed to this project. It has also been found that MKV container stores the cluster's data in hexadecimal format, therefore, it has been contributed that conversion of hidden data to hexadecimal format needs to take place before inserting any data inside the MKV container. Semi-functional encryption algorithm has also been contributed, which shows the possibility of using any encryption algorithm in order to increase the safety of the hidden data. The functional data embedding algorithm has been achieved, and finally, the data detecting algorithm, which allows users to verify the integrity of their hidden data in MKV container, has also been contributed.

6.3. Future works and Directions

Since data embedding has been successfully achieved, and data detecting has also been contributed in this project, it would be logical to develop a data removing or even data modifying algorithm. The algorithm needs to be developed the way so that it is able to completely remove the hidden data or even modify the data directly from the program just by simply entering the encryption key and the cluster's timecode. Because users tend to make mistakes, such as entering the wrong text or even entering the wrong encryption key, and mistakes in data hiding are not acceptable, this algorithm will play significant role in reversing all the unsatisfactory actions.

It would also be interesting to look at other file formats like, images, and find a way to hide the data inside there.

References

Bornaghi, C., & Damiani, E. (2008). Using Matroska EBML for Right Management. In International Conference on Automated Solution for Cross Media Content and Multi-channel distribution (AXMEDIS)-Workshop on Digital Preservation Weaving Factory (pp. 51-56). Firenze university press.

Cheng, X., Dale, C., & Liu, J. (2008, June). Statistics and social network of youtube videos. In Quality of Service, 2008. IWQoS 2008. 16th International Workshop on (pp. 229-238). IEEE.

Download Sample Videos. (2017). Retrieved October 17, 2017, from Sample Videos: http://www.sample-videos.com/index.php#sample-mkv-video

Diagram. (2016). Retrieved August 24, 2017, from Matroska: https://matroska.org/technical/diagram/index.html#detailed

Gordon, W. (2012, March 14). What's the Difference Between All These Video Formats, and Which One Should I Use? Retrieved August 9, 2017, from LifeHacker: http://lifehacker.com/5893250/whats-the-difference-between-all-these-video-formats-and-which-one-should-i-use

Lunden, I. (2015, June 2). 6.1B smartphone users globally by 2020, overtaking basic fixed phone subscriptions. Retrieved August 9, 2017, from Tech Crunch: https://techcrunch.com/2015/06/02/6-1b-smartphone-users-globally-by-2020-overtakingbasic-fixed-phone-subscriptions/

Matroska FAQ. (2017). Retrieved August 10, 2017, from Matroska: https://www.matroska.org/technical/guides/faq/index.html

Maung, I. M., Tew, Y., & Wong, K. (2016, May). Authentication for aac compressed audio using data hiding. In Consumer Electronics-Taiwan (ICCE-TW), 2016 IEEE International Conference on (pp. 1-2). IEEE.

Navas, K. A., Vidya, V., & Dass, S. V. (2011, September). High security data embedding in video. In *Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE* (pp. 647-651). IEEE.

Noé, A. (2009). Matroska File Format (under construction!). *Jun*, *24*, 1-51.

Orlin, J. (2012). Survey: MP4 is top format for web and mobile videos. Retrieved August 9, 2017, from Tech Crunch: https://techcrunch.com/2012/04/17/survey-mp4-is-top-format-for-web-and-mobile-videos/

Shukela, V. (2012). Simple Python Matroska (mkv) reading library, also mkv2xml and xml2mkv. Retrieved October 17, 2017, from GitHub: https://github.com/vi/mkvparse

Smith, C. (2016, December 23). MKV vs MP4, The Differences Between MKV and MP4. Retrieved August 10, 2017, from Keepvid: https://pro.keepvid.com/mp4/mkv-vs-mp4.html

Specifications (2017). Retrieved October 16, 2017, from Matroska: https://matroska.org/technical/specs/index.html#Cluster

Tew, Y., Wong, K., Phan, R. C. W., & Ngan, K. N. (2016). Multi-layer authentication scheme for HEVC video based on embedded statistics. Journal of Visual Communication and Image Representation, 40, 502-515.

Tsai, C. L., Fan, K. C., & Chung, C. D. (2001, October). Secure information by using digital data embedding and spread spectrum techniques. In *Security Technology, 2001 IEEE 35th International Carnahan Conference on* (pp. 156-162). IEEE.

Appendix

Word Count (without citations): 5,700

Test Plan

1. Introduction

Since the functionality of the MKV file is the first priority, during each phase of coding, the MKV file will need to be tested whether it is functional. In order to identify the functionality of the file, the MKV containers needs to be executed and checked whether it has any distortions (both audio and video). Three files of varied sizes, such as 2 MB (720p), 21 MB (720p), and 137MB (4K), have been selected in order to ensure that data hiding, and detecting algorithms are able to hide and detect the data inside the file with completely different video length and as well as different file sizes.

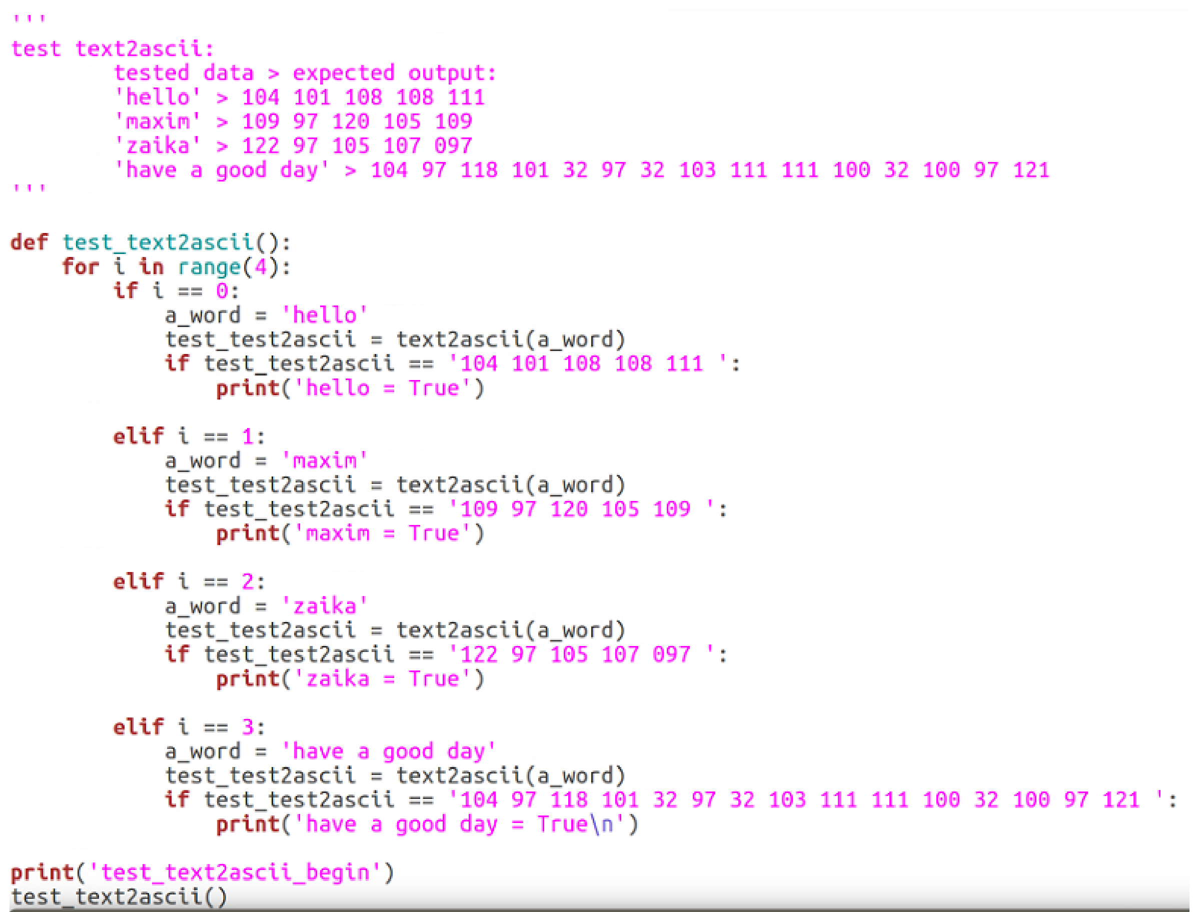
2. Test Report

2.1. Unit Testing (for each module)

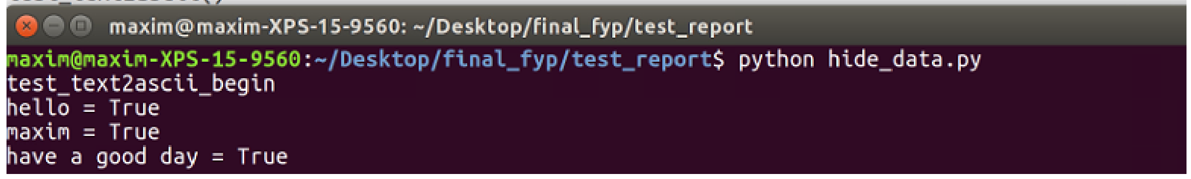
Screenshots of testings are provided for all the Functions in Data Hiding & Data Embedding Algorithms. Text2ascii, pumpKey, encryption algorithm, convert2hex have been tested using the expected output.

2.1.1. Text2ascii Function

This function converts text to ascii with all the required spaces. Spaces are later converted using hexadecimal values (convert2hex function). Converted values are checked online using the text to ascii converter. The testing performed based on online results. Output is provided, and the expected output for all the values is “True”. The testing of the wrong output is not required. Testings of the function are displayed below:



```
'''
test text2ascii:
        tested data > expected output:
        'hello' > 104 101 108 108 111
        'maxim' > 109 97 120 105 109
        'zaika' > 122 97 105 107 097
        'have a good day' > 104 97 118 101 32 97 32 103 111 111 100 32 100 97 121
'''

def test_text2ascii():
    for i in range(4):
        if i == 0:
            a_word = 'hello'
            test_test2ascii = text2ascii(a_word)
            if test_test2ascii == '104 101 108 108 111 ':
                print('hello = True')

        elif i == 1:
            a_word = 'maxim'
            test_test2ascii = text2ascii(a_word)
            if test_test2ascii == '109 97 120 105 109 ':
                print('maxim = True')

        elif i == 2:
            a_word = 'zaika'
            test_test2ascii = text2ascii(a_word)
            if test_test2ascii == '122 97 105 107 097 ':
                print('zaika = True')

        elif i == 3:
            a_word = 'have a good day'
            test_test2ascii = text2ascii(a_word)
            if test_test2ascii == '104 97 118 101 32 97 32 103 111 111 100 32 100 97 121 ':
                print('have a good day = True\n')

print('test_text2ascii_begin')
test_text2ascii()
```

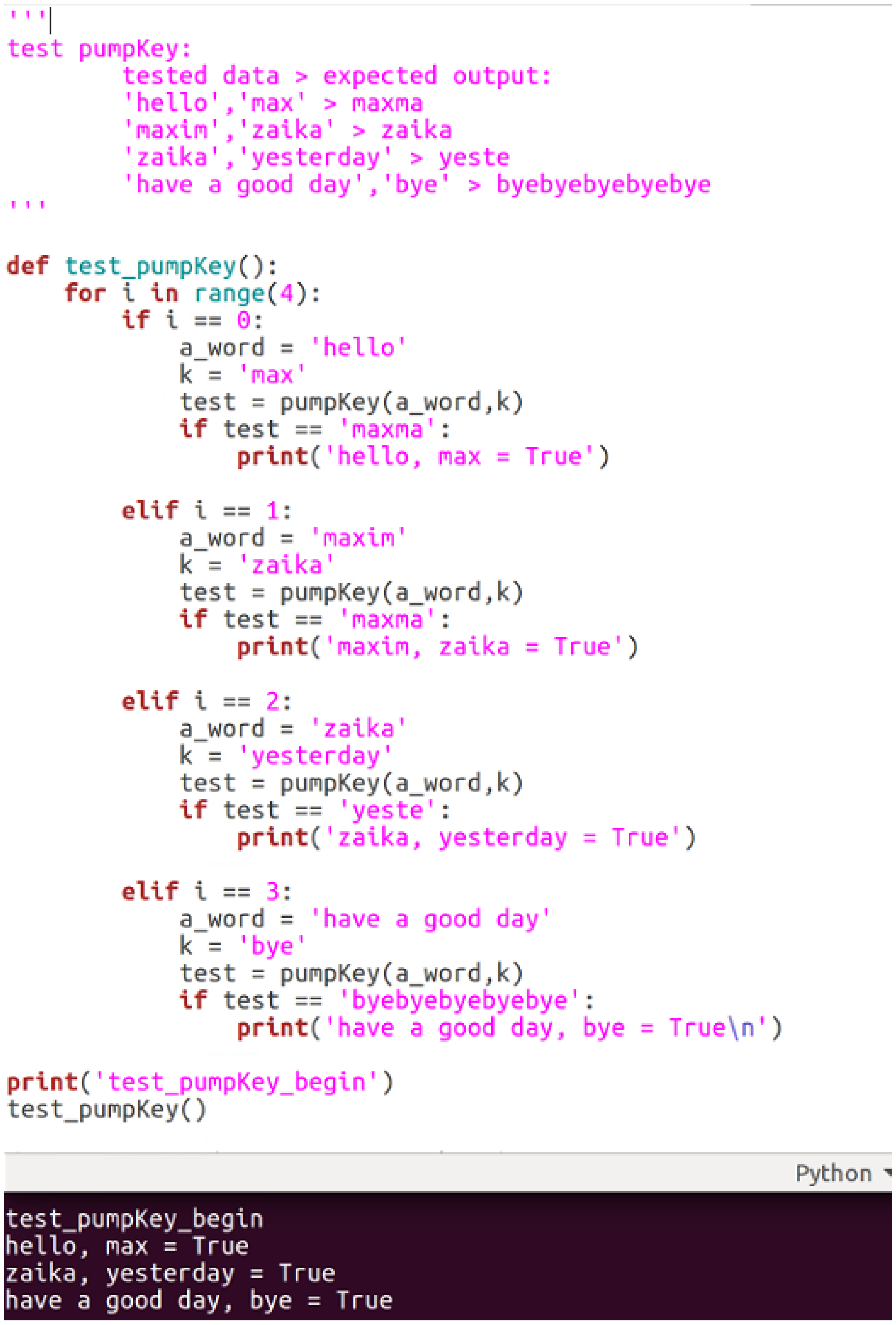


```
maxim@maxim-XPS-15-9560: ~/Desktop/final_fyp/test_report
maxim@maxim-XPS-15-9560:~/Desktop/final_fyp/test_report$ python hide_data.py
test_text2ascii_begin
hello = True
maxim = True
have a good day = True
```

Therefore, function performs all the actions correctly.

2.1.2. pumpKey Function

In this function whenever the size (the length) of the encryption key is less than the length of the text, key size increases to match text size by looping itself. Output is provided, and the expected output for all the values is “True”. The testing of the wrong output is not required. Testings of the function are displayed below:



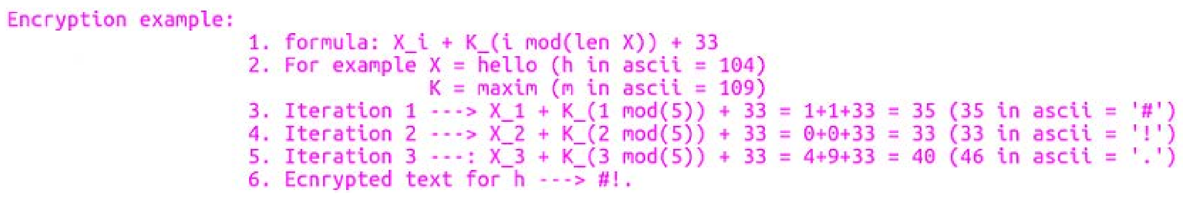
```
'''
test pumpKey:
        tested data > expected output:
        'hello','max' > maxma
        'maxim','zaika' > zaika
        'zaika','yesterday' > yeste
        'have a good day','bye' > byebyebyebyebye
'''

def test_pumpKey():
    for i in range(4):
        if i == 0:
            a_word = 'hello'
            k = 'max'
            test = pumpKey(a_word,k)
            if test == 'maxma':
                print('hello, max = True')

        elif i == 1:
            a_word = 'maxim'
            k = 'zaika'
            test = pumpKey(a_word,k)
            if test == 'maxma':
                print('maxim, zaika = True')

        elif i == 2:
            a_word = 'zaika'
            k = 'yesterday'
            test = pumpKey(a_word,k)
            if test == 'yeste':
                print('zaika, yesterday = True')

        elif i == 3:
            a_word = 'have a good day'
            k = 'bye'
            test = pumpKey(a_word,k)
            if test == 'byebyebyebyebye':
                print('have a good day, bye = True\n')

print('test_pumpKey_begin')
test_pumpKey()
```

```
test_pumpKey_begin
hello, max = True
zaika, yesterday = True
have a good day, bye = True
```

Therefore, function performs all the actions correctly.

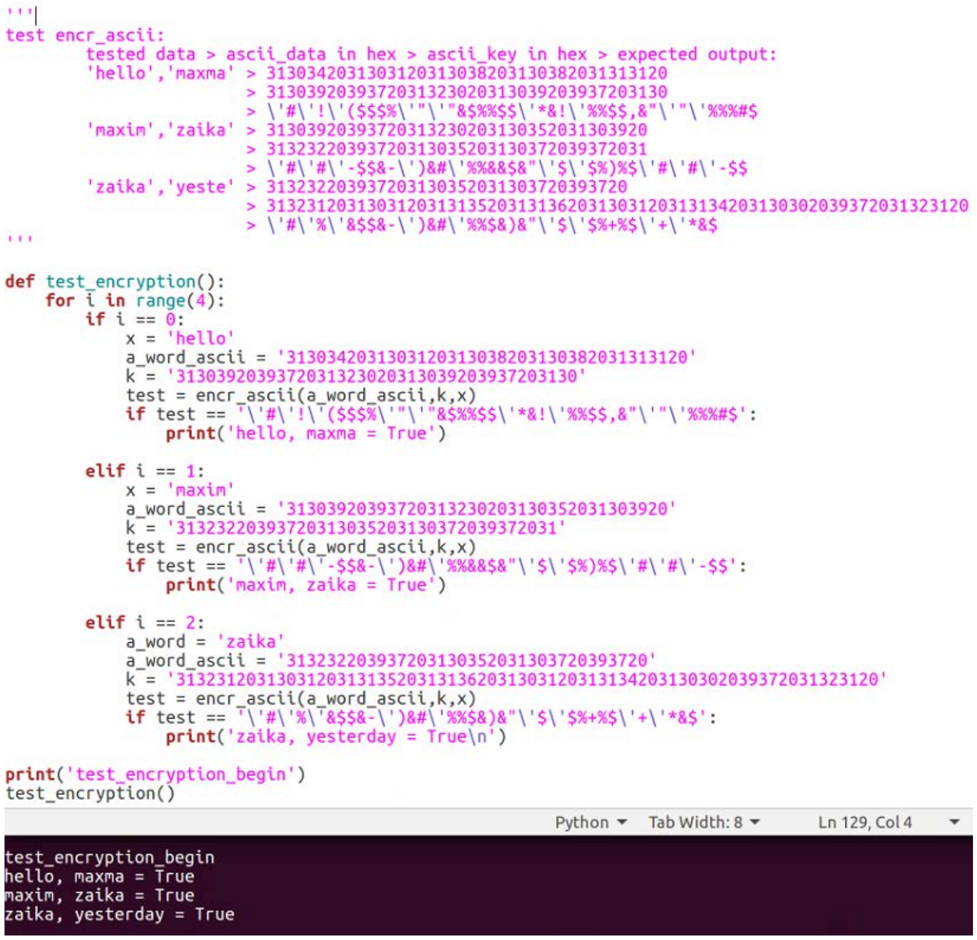
2.1.3. encr_ascii Function

This function accepts the “text” in hexadecimal format, the “key” in hexadecimal format, and the original “text.” Conversion of the hexadecimal values have been taken from the online source to ensure that the output is correct. Example of the encryption algorithm has been explained in the final report; however, proper examples are provided below in the screenshot form:



```
Encryption example:
                         1. formula: X_i + K_(i mod(len X)) + 33
                         2. For example X = hello (h in ascii = 104)
                                        K = maxim (m in ascii = 109)
                         3. Iteration 1 ---> X_1 + K_(1 mod(5)) + 33 = 1+1+33 = 35 (35 in ascii = '#')
                         4. Iteration 2 ---> X_2 + K_(2 mod(5)) + 33 = 0+0+33 = 33 (33 in ascii = '!')
                         5. Iteration 3 ---: X_3 + K_(3 mod(5)) + 33 = 4+9+33 = 40 (46 in ascii = '.')
                         6. Ecnrypted text for h ---> #!.
```

Expected output and the actual outputs are provided below in the form of a sreenshot.



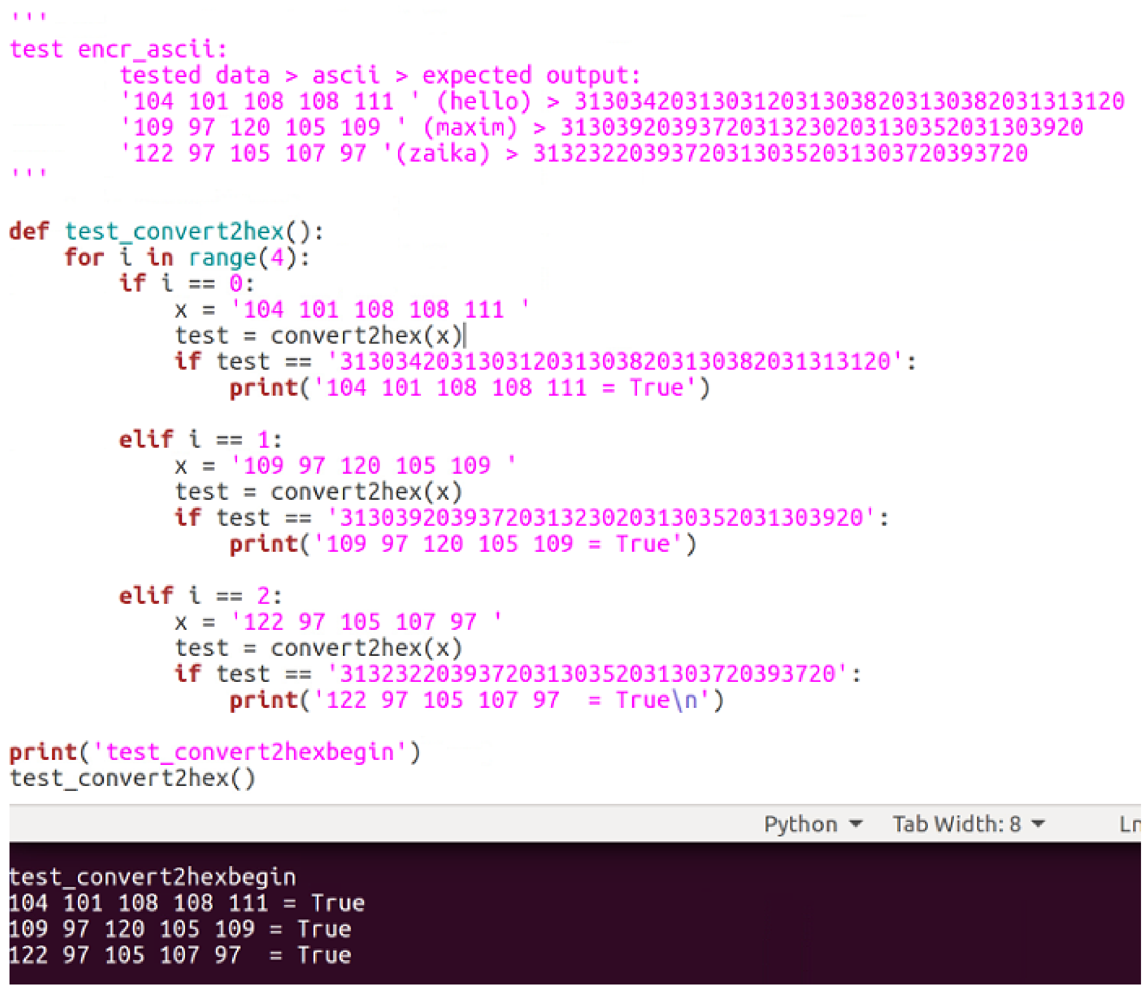
```
'''
test encr_ascii:
        tested data > ascii_data in hex > ascii_key in hex > expected output:
        'hello','maxma' > 3130342031303120313038203130382031313120
                        > 3130392039372031323020313039203937203130
                        > \'#\'!\'($$$%\'"\'"&$%%$$\'*&!\'%%$$,&"\'"\'%%%#$
        'maxim','zaika' > 31303920393720313230203130352031303920
                        > 31323220393720313035203130372039372031
                        > \'#\'#\'-$$&-\')&#\'%%&&$&"\'$\'$%)%$\'#\'#\'-$$
        'zaika','yeste' > 313232203937203130352031303720393720
                        > 3132312031303120313135203131362031303120313134203130302039372031323120
                        > \'#\'%\'&$$&-\')&#\'%%$&)&"\'$\'$%+%$\'+\'*&$
'''

def test_encryption():
    for i in range(4):
        if i == 0:
            x = 'hello'
            a_word_ascii = '3130342031303120313038203130382031313120'
            k = '3130392039372031323020313039203937203130'
            test = encr_ascii(a_word_ascii,k,x)
            if test == '\'#\'!\'($$$%\'"\'"&$%%$$\'*&!\'%%$$,&"\'"\'%%%#$':
                print('hello, maxma = True')

        elif i == 1:
            x = 'maxim'
            a_word_ascii = '31303920393720313230203130352031303920'
            k = '31323220393720313035203130372039372031'
            test = encr_ascii(a_word_ascii,k,x)
            if test == '\'#\'#\'-$$&-\')&#\'%%&&$&"\'$\'$%)%$\'#\'#\'-$$':
                print('maxim, zaika = True')

        elif i == 2:
            a_word = 'zaika'
            a_word_ascii = '313232203937203130352031303720393720'
            k = '3132312031303120313135203131362031303120313134203130302039372031323120'
            test = encr_ascii(a_word_ascii,k,x)
            if test == '\'#\'%\'&$$&-\')&#\'%%$&)&"\'$\'$%+%$\'+\'*&$':
                print('zaika, yesterday = True\n')

print('test_encryption_begin')
test_encryption()
```

```
test_encryption_begin
hello, maxma = True
maxim, zaika = True
zaika, yesterday = True
```

Therefore, function performs all the actions correctly.

2.1.4. convert2hex Function:

This function accepts the original test in ascii format and coverts it to hexadecimal format because MKV acceptes hexadecimal data. In order to ensure that ascii and hexadecimal values are correct, online source has been used and compared with my output. The expect and the actual output are provided below in the screenshot form:



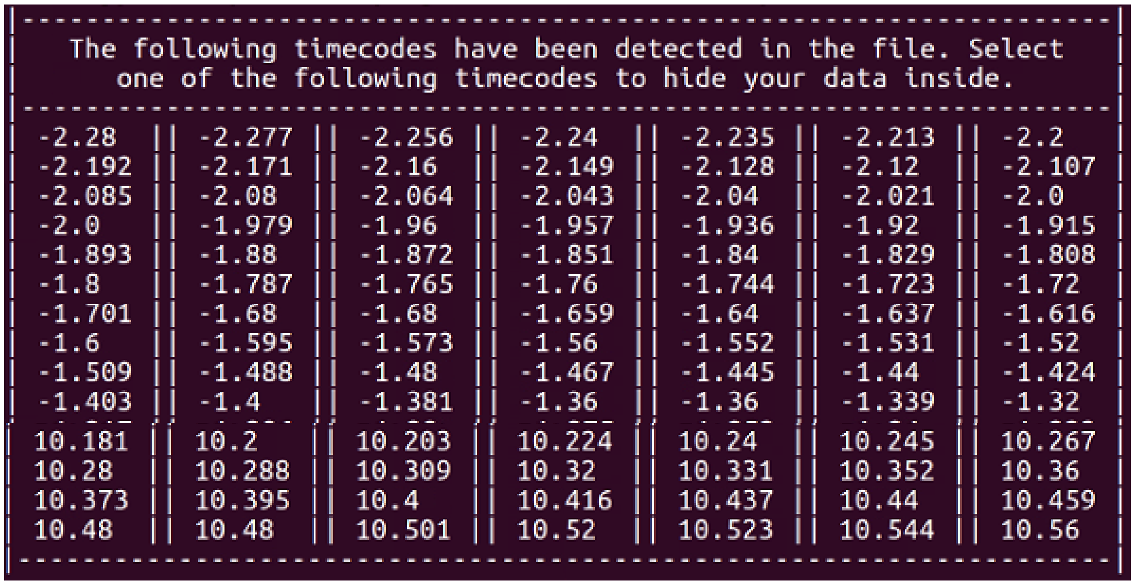
```
'''
test encr_ascii:
        tested data > ascii > expected output:
        '104 101 108 108 111 ' (hello) > 3130342031303120313038203130382031313120
        '109 97 120 105 109 ' (maxim) > 31303920393720313230203130352031303920
        '122 97 105 107 97 '(zaika) > 313232203937203130352031303720393720
'''

def test_convert2hex():
    for i in range(4):
        if i == 0:
            x = '104 101 108 108 111 '
            test = convert2hex(x)
            if test == '3130342031303120313038203130382031313120':
                print('104 101 108 108 111 = True')

        elif i == 1:
            x = '109 97 120 105 109 '
            test = convert2hex(x)
            if test == '31303920393720313230203130352031303920':
                print('109 97 120 105 109 = True')

        elif i == 2:
            x = '122 97 105 107 97 '
            test = convert2hex(x)
            if test == '313232203937203130352031303720393720':
                print('122 97 105 107 97  = True\n')

print('test_convert2hexbegin')
test_convert2hex()
```

```
test_convert2hexbegin
104 101 108 108 111 = True
109 97 120 105 109 = True
122 97 105 107 97  = True
```

Therefore, function performs all the actions correctly.

2.1.5. Get_timecodes and return_timecodes Functions:

The testing of these functions is very tricky and almost impossible for the large files unless the checking has been performed manually. Get_timecodes function retrieves and stores all the timecodes from the files, while Return_Timecodes function prints all the the timecodes in the right order. Since the testing is impossible, the testing has been performed manually on the small file. Even though the file is small, it is still impossible to attach the screenshot because it will take up to 10 pages. Therefore, the testing of thefunction return_timecodes have been performed, where it prints out all the timecodes of the MKV file by typing 'help' in the code itself. Cropped version of the screenshot has been attached below:

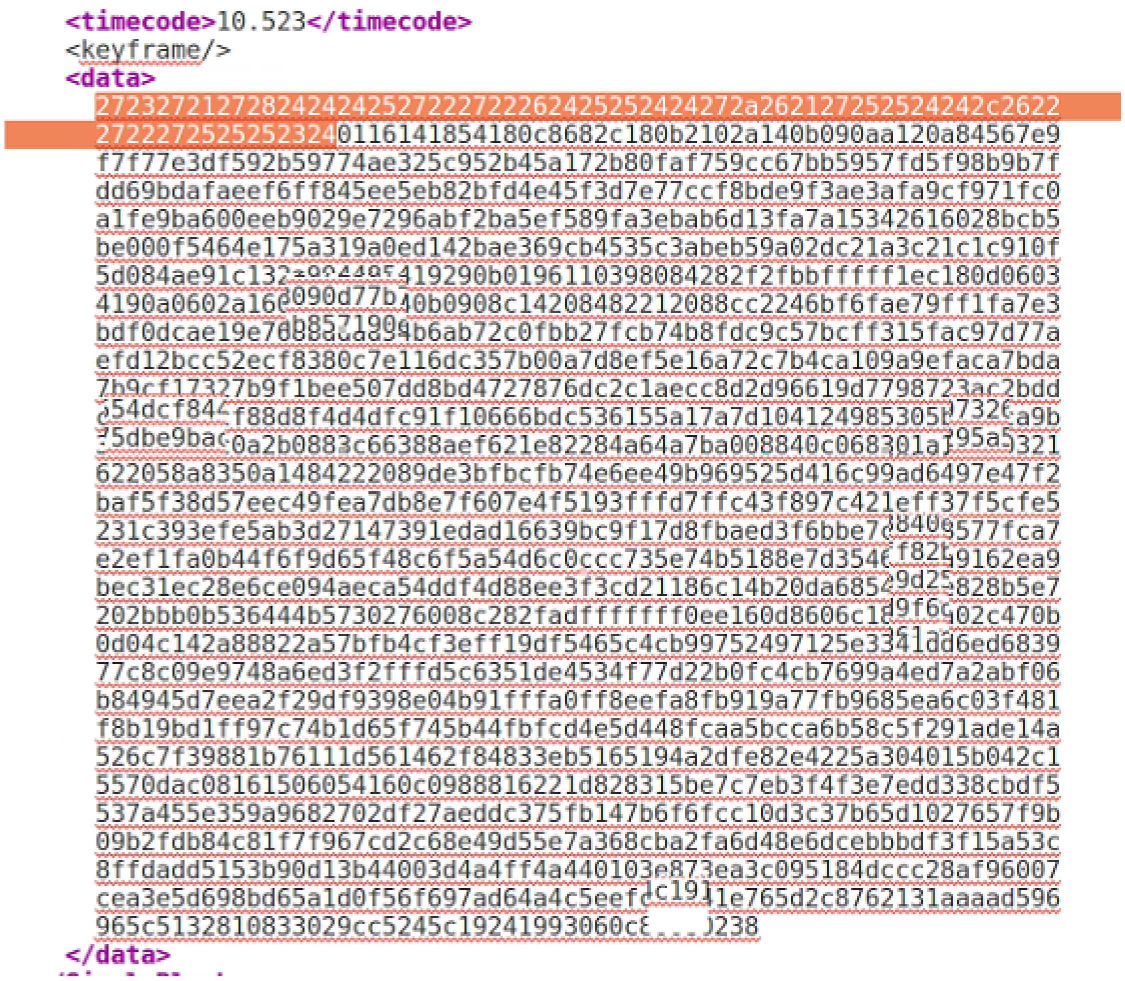


```
|-------------------------------------------------------------|
|   The following timecodes have been detected in the file. Select   |
|      one of the following timecodes to hide your data inside.      |
|-------------------------------------------------------------|
| -2.28  || -2.277 || -2.256 || -2.24  || -2.235 || -2.213 || -2.2   |
| -2.192 || -2.171 || -2.16  || -2.149 || -2.128 || -2.12  || -2.107 |
| -2.085 || -2.08  || -2.064 || -2.043 || -2.04  || -2.021 || -2.0   |
| -2.0   || -1.979 || -1.96  || -1.957 || -1.936 || -1.92  || -1.915 |
| -1.893 || -1.88  || -1.872 || -1.851 || -1.84  || -1.829 || -1.808 |
| -1.8   || -1.787 || -1.765 || -1.76  || -1.744 || -1.723 || -1.72  |
| -1.701 || -1.68  || -1.68  || -1.659 || -1.64  || -1.637 || -1.616 |
| -1.6   || -1.595 || -1.573 || -1.56  || -1.552 || -1.531 || -1.52  |
| -1.509 || -1.488 || -1.48  || -1.467 || -1.445 || -1.44  || -1.424 |
| -1.403 || -1.4   || -1.381 || -1.36  || -1.36  || -1.339 || -1.32  |
| 10.181 || 10.2   || 10.203 || 10.224 || 10.24  || 10.245 || 10.267 |
| 10.28  || 10.288 || 10.309 || 10.32  || 10.331 || 10.352 || 10.36  |
| 10.373 || 10.395 || 10.4   || 10.416 || 10.437 || 10.44  || 10.459 |
| 10.48  || 10.48  || 10.501 || 10.52  || 10.523 || 10.544 || 10.56  |
|-------------------------------------------------------------|
```

The above version is cropped because there are way too many timecode; however, the output is clear and allows users to select any desired timecode.

2.1.6. Hide_data Function:

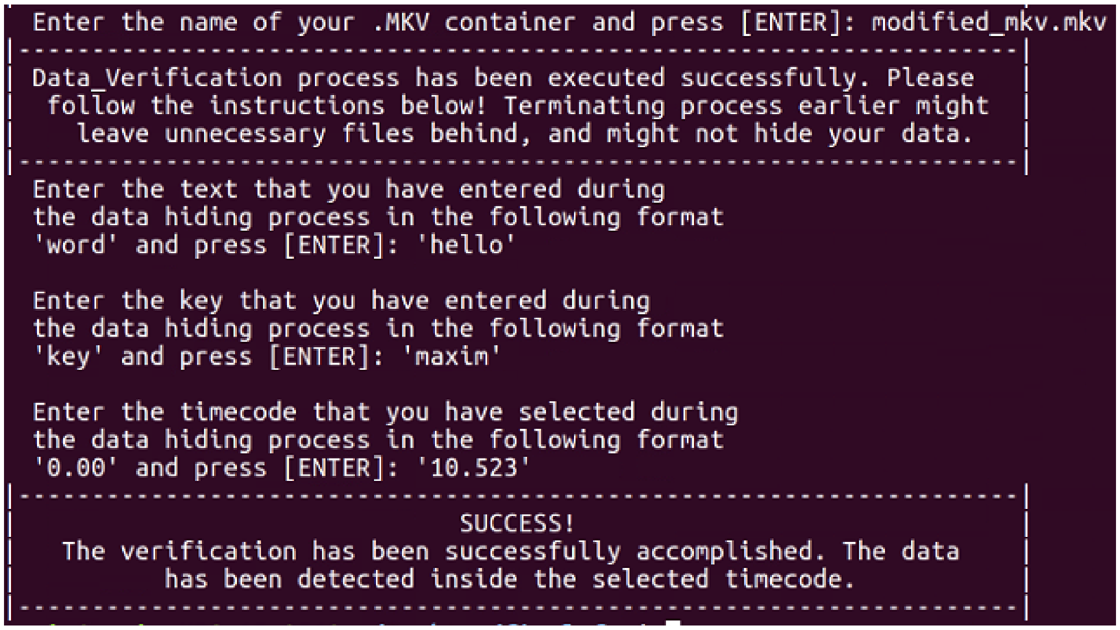
It is impossible to check whether this function works or not because it always works of this function; however, the screenshot of hidden data is attached below. Assume that the hidden text is 'hello' (hexadecimal encrypted word is **272327212728242424252722272226242525252424272a262127252524242c262227222272525252324** for the key 'maxim'), and timecode is '10.523' therefore the hidden data needs to be as described. Expected output is below. Screenshot of hidden data is attached below:



```
<timecode>10.523</timecode>
<keyframe/>
<data>
  27232721272824242425272227222624252524242 72a262127252524242c2622
  2722272525252324011614185418 0c8682c180b2102a140b090aa120a84567e9
  f7f77e3df592b59774ae325c952b45a172b80faf759cc67bb5957fd5f98b9b7f
  dd69bdafaeef6ff845ee5eb82bfd4e45f3d7e77ccf8bde9f3ae3afa9cf971fc0
  a1fe9ba600eeb9029e7296abf2ba5ef589fa3ebab6d13fa7a15342616028bcb5
  be000f5464e175a319a0ed142bae369cb4535c3abeb59a02dc21a3c21c1c910f
  5d084ae91c132[illegible]419290b0196110398084282f2fbbfffff1ec180d0603
  4190a0602a16[illegible]40b0908c14208482212088cc2246bf6fae79ff1fa7e3
  bdf0dcae19e7[illegible]4b6ab72c0fbb27fcb74b8fdc9c57bcff315fac97d77a
  efd12bcc52ecf8380c7e116dc357b00a7d8ef5e16a72c7b4ca109a9efaca7bda
  7b9cf17327b9f1bee507dd8bd4727876dc2c1aecc8d2d96619d779872 3ac2bdd
  [illegible]54dcf844f88d8f4d4dfc91f10666bdc536155a17a7d104124985305[illegible]a9b
  [illegible]5dbe9ba[illegible]0a2b0883c66388aef621e82284a64a7ba008840c06830 1a[illegible]321
  622058a8350a1484222089de3bfbcfb74e6ee49b969525d416c99ad6497e47f2
  baf5f38d57eec49fea7db8e7f607e4f5193fffd7ffc43f897c421eff37f5cfe5
  231c393efe5ab3d27147391edad16639bc9f17d8fbaed3f6bbe7[illegible]577fca7
  e2ef1fa0b44f6f9d65f48c6f5a54d6c0ccc735e74b5188e7d354[illegible]9162ea9
  bec31ec28e6ce094aeca54ddf4d88ee3f3cd21186c14b20da685[illegible]828b5e7
  202bbb0b536444b5730276008c282fadffffff0ee160d8606c1[illegible]02c470b
  0d04c142a88822a57bfb4cf3eff19df5465c4cb99752497125e3341dd6ed6839
  77c8c09e9748a6ed3f2fffd5c6351de4534f77d22b0fc4cb7699a4ed7a2abf06
  b84945d7eea2f29df9398e04b91fffa0ff8eefa8fb919a77fb9685ea6c03f481
  f8b19bd1ff97c74b1d65f745b44fbfcd4e5d448fcaa5bcca6b58c5f291ade14a
  526c7f39881b76111d561462f84833eb5165194a2dfe82e4225a304015b042c1
  5570dac08161506054160c0988816221d828315be7c7eb3f4f3e7edd338cbdf5
  537a455e359a9682702df27aeddc375fb147b6f6fcc10d3c37b65d1027657f9b
  09b2fdb84c81f7f967cd2c68e49d55e7a368cba2fa6d48e6dcebbbdf3f15a53c
  8ffdadd5153b90d13b44003d4a4ff4a440103e873ea3c095184dccc28af96007
  cea3e5d698bd65a1d0f56f697ad64a4c5eef[illegible]1e765d2c8762131aaaad596
  965c5132810833029cc5245c19241993060c[illegible]238
</data>
```

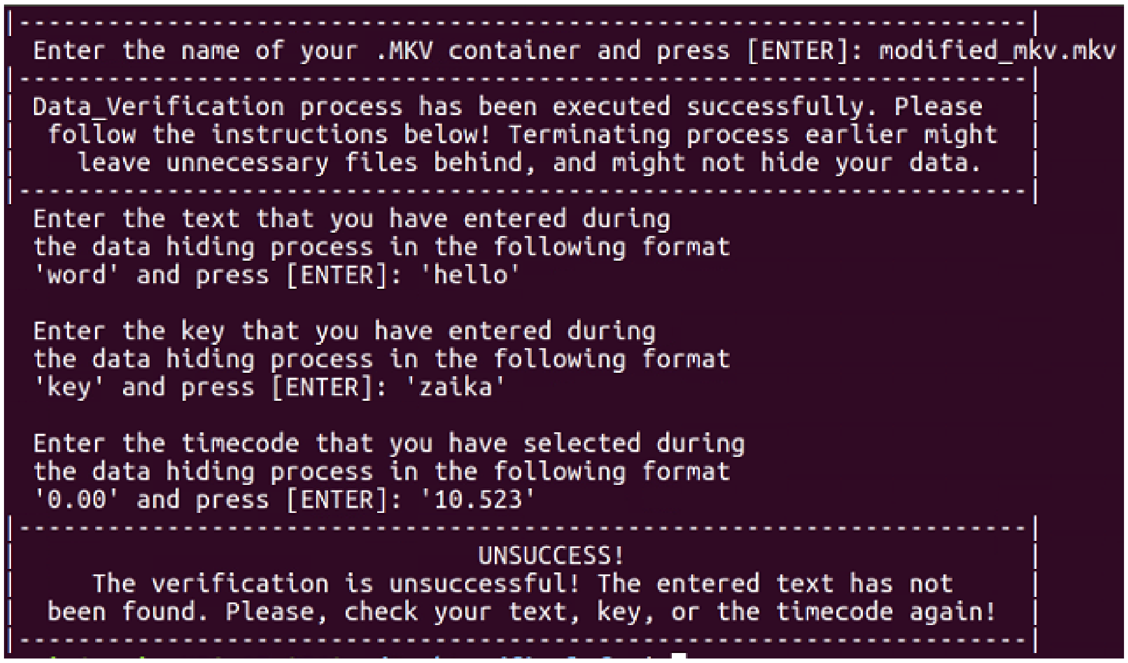
2.1.7. Find_data Function

There is no way to check in the code way whether the output of the function is valid. However, some examples can be shown. Following the example from the hide_data function, let’s reverse the process on the modified file entering exactly the same data and find out whether the data is actually hidden in the container. Expected output is “success”, screenshot is attached below:



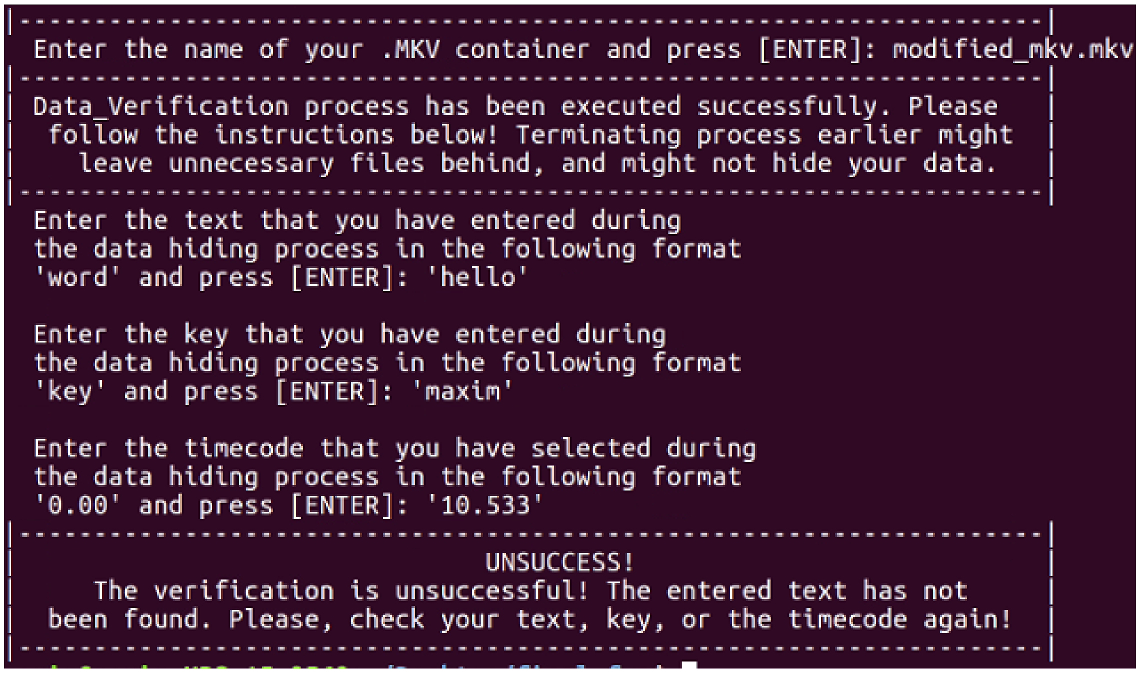
```
 Enter the name of your .MKV container and press [ENTER]: modified_mkv.mkv
|--------------------------------------------------------------------|
| Data_Verification process has been executed successfully. Please   |
|  follow the instructions below! Terminating process earlier might  |
|    leave unnecessary files behind, and might not hide your data.   |
|--------------------------------------------------------------------|
 Enter the text that you have entered during
 the data hiding process in the following format
 'word' and press [ENTER]: 'hello'

 Enter the key that you have entered during
 the data hiding process in the following format
 'key' and press [ENTER]: 'maxim'

 Enter the timecode that you have selected during
 the data hiding process in the following format
 '0.00' and press [ENTER]: '10.523'
|--------------------------------------------------------------------|
|                              SUCCESS!                              |
|   The verification has been successfully accomplished. The data    |
|          has been detected inside the selected timecode.           |
|--------------------------------------------------------------------|
```

Let’s try to input the wrong key, for example ‘zaika’. The expected output is “unsuccess”, screenshot is attached below:



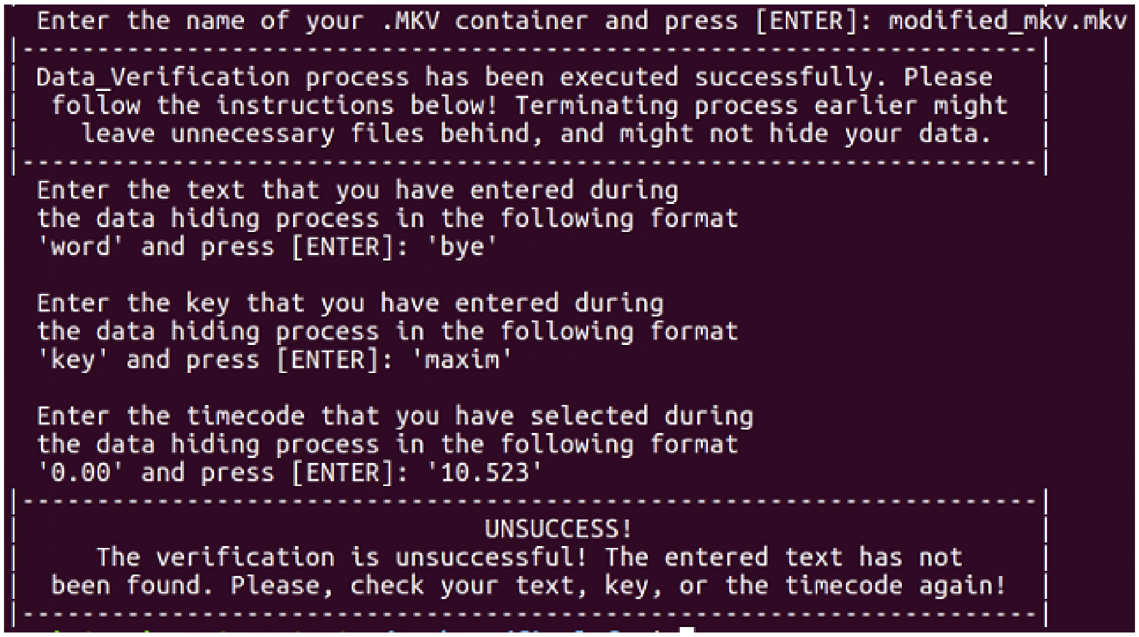
```
|--------------------------------------------------------------------|
 Enter the name of your .MKV container and press [ENTER]: modified_mkv.mkv
|--------------------------------------------------------------------|
| Data_Verification process has been executed successfully. Please   |
|  follow the instructions below! Terminating process earlier might  |
|    leave unnecessary files behind, and might not hide your data.   |
|--------------------------------------------------------------------|
 Enter the text that you have entered during
 the data hiding process in the following format
 'word' and press [ENTER]: 'hello'

 Enter the key that you have entered during
 the data hiding process in the following format
 'key' and press [ENTER]: 'zaika'

 Enter the timecode that you have selected during
 the data hiding process in the following format
 '0.00' and press [ENTER]: '10.523'
|--------------------------------------------------------------------|
|                             UNSUCCESS!                             |
|     The verification is unsuccessful! The entered text has not     |
|  been found. Please, check your text, key, or the timecode again!  |
|--------------------------------------------------------------------|
```

Same output is expected for the wrong timecode, for example ’10.533’. The expected output is “unsuccess”, screenshot is attached below:



```
|-----------------------------------------------------------------|
  Enter the name of your .MKV container and press [ENTER]: modified_mkv.mkv
|-----------------------------------------------------------------|
| Data_Verification process has been executed successfully. Please |
|  follow the instructions below! Terminating process earlier might |
|    leave unnecessary files behind, and might not hide your data.  |
|-----------------------------------------------------------------|
  Enter the text that you have entered during
  the data hiding process in the following format
  'word' and press [ENTER]: 'hello'

  Enter the key that you have entered during
  the data hiding process in the following format
  'key' and press [ENTER]: 'maxim'

  Enter the timecode that you have selected during
  the data hiding process in the following format
  '0.00' and press [ENTER]: '10.533'
|-----------------------------------------------------------------|
|                            UNSUCCESS!                            |
|     The verification is unsuccessful! The entered text has not   |
|  been found. Please, check your text, key, or the timecode again! |
|-----------------------------------------------------------------|
```

Same out is expected for the wrong text, for example ‘bye’. The expected output is ‘unsuccess’, screenshot is attached below:



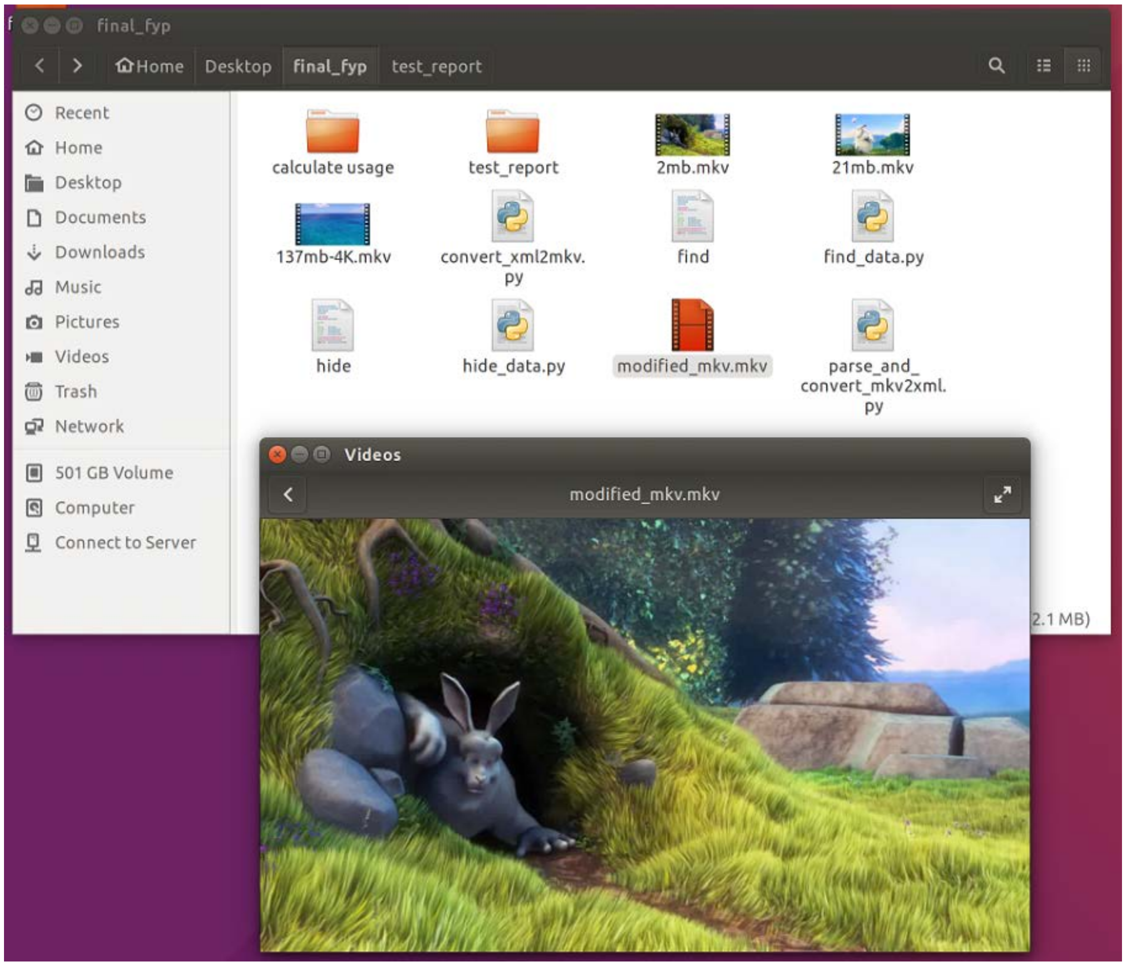
```
  Enter the name of your .MKV container and press [ENTER]: modified_mkv.mkv
|-----------------------------------------------------------------|
| Data_Verification process has been executed successfully. Please |
|  follow the instructions below! Terminating process earlier might |
|    leave unnecessary files behind, and might not hide your data.  |
|-----------------------------------------------------------------|
  Enter the text that you have entered during
  the data hiding process in the following format
  'word' and press [ENTER]: 'bye'

  Enter the key that you have entered during
  the data hiding process in the following format
  'key' and press [ENTER]: 'maxim'

  Enter the timecode that you have selected during
  the data hiding process in the following format
  '0.00' and press [ENTER]: '10.523'
|-----------------------------------------------------------------|
|                            UNSUCCESS!                            |
|     The verification is unsuccessful! The entered text has not   |
|  been found. Please, check your text, key, or the timecode again! |
|-----------------------------------------------------------------|
```

Since all the executions return the correct output, the function functions correctly.

2.2. System Testing

2.2.1. Integration Testing

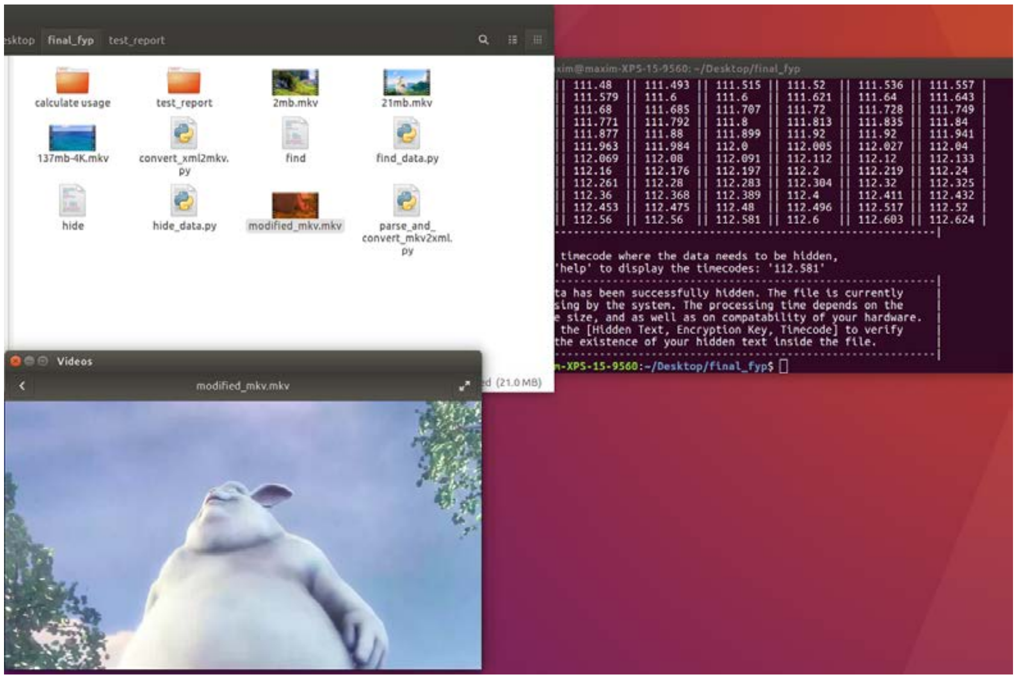
Whenever the data gets hidden in the MKV container, the edited MKV container is expected to execute without any distortions. Testing is performed with exactly the same data as has been entered in the unit testing 2.1.6., and 2 MB file. The algorithm is expected to return modified_mkv.mkv file without absolutely any distortions; however, it is expected not to have a preview image because sometimes system restart is required, or another attempt is required.



Above file is perfectly functional.

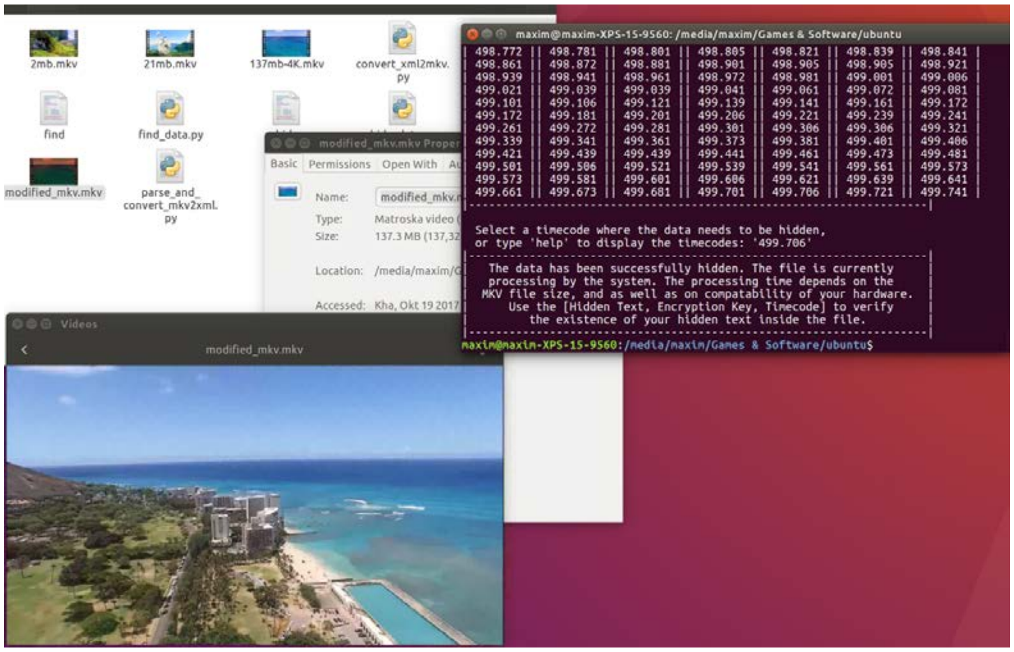
2.2.2. Stress Testing

During this testing, the testing has been performed on 21 MB and 137 MB (in 4K quality) to ensure that output is functional and without any distortions.



```
xim@maxim-XPS-15-9560: ~/Desktop/final_fyp
|| 111.48  || 111.493 || 111.515 || 111.52  || 111.536 || 111.557 |
|| 111.579 || 111.6   || 111.6   || 111.621 || 111.64  || 111.643 |
|| 111.68  || 111.685 || 111.707 || 111.72  || 111.728 || 111.749 |
|| 111.771 || 111.792 || 111.8   || 111.813 || 111.835 || 111.84  |
|| 111.877 || 111.88  || 111.899 || 111.92  || 111.92  || 111.941 |
|| 111.963 || 111.984 || 112.0   || 112.005 || 112.027 || 112.04  |
|| 112.069 || 112.08  || 112.091 || 112.112 || 112.12  || 112.133 |
|| 112.16  || 112.176 || 112.197 || 112.2   || 112.219 || 112.24  |
|| 112.261 || 112.28  || 112.283 || 112.304 || 112.32  || 112.325 |
|| 112.36  || 112.368 || 112.389 || 112.4   || 112.411 || 112.432 |
|| 112.453 || 112.475 || 112.48  || 112.496 || 112.517 || 112.52  |
|| 112.56  || 112.56  || 112.581 || 112.6   || 112.603 || 112.624 |
------------------------------------------------------------------|

 timecode where the data needs to be hidden,
'help' to display the timecodes: '112.581'
------------------------------------------------------------------|
ta has been successfully hidden. The file is currently           |
sing by the system. The processing time depends on the           |
e size, and as well as on compatability of your hardware.        |
 the [Hidden Text, Encryption Key, Timecode] to verify           |
the existence of your hidden text inside the file.               |
------------------------------------------------------------------|
m-XPS-15-9560:~/Desktop/final_fyp$ 
```

Above 21 MB file is perfectly functional.



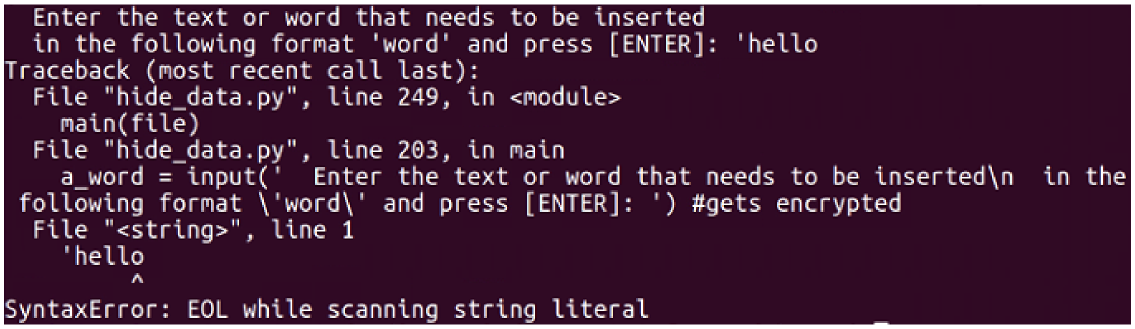
Name: modified_mkv.r
Type: Matroska video (
Size: 137.3 MB (137,32
Location: /media/maxim/G
Accessed: Kha, Okt 19 2017

```
maxim@maxim-XPS-15-9560: /media/maxim/Games & Software/ubuntu
| 498.772 || 498.781 || 498.801 || 498.805 || 498.821 || 498.839 || 498.841 |
| 498.861 || 498.872 || 498.881 || 498.901 || 498.905 || 498.905 || 498.921 |
| 498.939 || 498.941 || 498.961 || 498.972 || 498.981 || 499.001 || 499.006 |
| 499.021 || 499.039 || 499.039 || 499.041 || 499.061 || 499.072 || 499.081 |
| 499.101 || 499.106 || 499.121 || 499.139 || 499.141 || 499.161 || 499.172 |
| 499.172 || 499.181 || 499.201 || 499.206 || 499.221 || 499.239 || 499.241 |
| 499.261 || 499.272 || 499.281 || 499.301 || 499.306 || 499.306 || 499.321 |
| 499.339 || 499.341 || 499.361 || 499.373 || 499.381 || 499.401 || 499.406 |
| 499.421 || 499.439 || 499.439 || 499.441 || 499.461 || 499.473 || 499.481 |
| 499.501 || 499.506 || 499.521 || 499.539 || 499.541 || 499.561 || 499.573 |
| 499.573 || 499.581 || 499.601 || 499.606 || 499.621 || 499.639 || 499.641 |
| 499.661 || 499.673 || 499.681 || 499.701 || 499.706 || 499.721 || 499.741 |
|--------------------------------------------------------------------|

 Select a timecode where the data needs to be hidden,
 or type 'help' to display the timecodes: '499.706'
|--------------------------------------------------------------------|
|   The data has been successfully hidden. The file is currently     |
|   processing by the system. The processing time depends on the     |
|  MKV file size, and as well as on compatability of your hardware.  |
|     Use the [Hidden Text, Encryption Key, Timecode] to verify      |
|        the existence of your hidden text inside the file.          |
|--------------------------------------------------------------------|
maxim@maxim-XPS-15-9560:/media/maxim/Games & Software/ubuntu$
```

Above 137 MB file is perfectly functional.

2.2.3. Acceptance Testing

Supervisor has tested the program and was satisfied with the output; however, when someone else tested the program, it crashed. It happened because encryption key was not able to accept integers (digits), therefore, the code has been redesigned and after that no more issues faced. Whenever the input is wrong and not following the format, program is expected to crash. It happens because there wes no error catching done in the code; however, it is perfectly functional. Example is provided below:



```
  Enter the text or word that needs to be inserted
  in the following format 'word' and press [ENTER]: 'hello
Traceback (most recent call last):
  File "hide_data.py", line 249, in <module>
    main(file)
  File "hide_data.py", line 203, in main
    a_word = input('  Enter the text or word that needs to be inserted\n  in the
 following format \'word\' and press [ENTER]: ') #gets encrypted
  File "<string>", line 1
    'hello
         ^
SyntaxError: EOL while scanning string literal
```

Since output is expected, it is considered that function works correctly. It crashed due to incorrect input **'hello**, and tells the user that the input is wrong and needs to be **'hello'**.

2.2.4. System Testing

Previously, it was expected for the code to work on all systems; however, by the end of the project, the code is able to be executed on Linux machines (Ubuntu or Macintosh) only.

3. Conclusion

From all the above tests, it has been concluded that system functions correctly; however, someone improvements need to be done, for example: error catching to make it more user friendly.

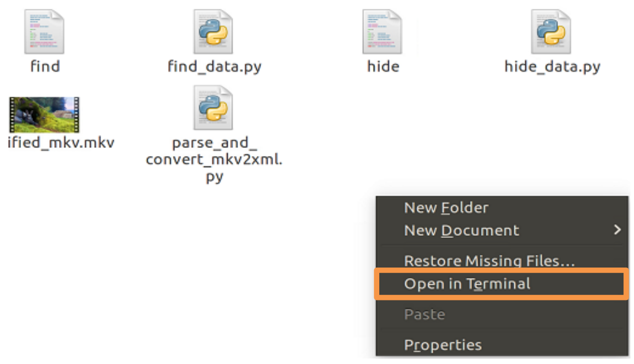
Run/Install/Compile Software

Important / Required Files

1. Access the Google Drive where the following files are located: 2mb.mkv, 21mb.mkv, 137mb-4K.mkv, convert_xml2mkv.py, find, find_data.py, hide, hide_data.py, parse_and_convert_mkv2xml.py, weekly report, and final report (The access will be granted upon request). Download **ALL** the files, except the reports (reports can be downloaded if necessary):

   https://drive.google.com/open?id=0B2tYPdcylmXYRG9TdkJibjRDa28

2. Launch Unix machine (Ubuntu or Macintosh), make sure that python is installed.
3. Download from drive and Copy all the files to a single folder, named, let's say "data_hiding_data_detecting"
   a. Folder's Name does not matter
   b. IMPORTANT TO HAVE ALL THE FILES INSIDE THE FOLDER
4. Launch the terminal from the folder.
   a. On ubuntu machine:
      i. Right click on the empty space and select "Open in Terminal" (if this feature is missing, make sure that Terminal is running from this folder)



5. To ensure that Ubuntu does not ask any permissions type the following commands in the terminal:

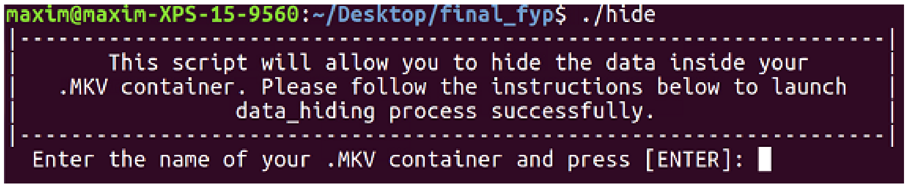
a. chmod +x convert_xml2mkv.py && chmod +x find_data.py && chmod +x hide_data.py && chmod +x parse_and_convert_mkv2xml.py

b. If system still asks the permission on other files that are not listed above like MKV container, please type the following command in the terminal:

   i. **chmod +x <u>filename.extension</u>**

<u>Data Embedding Process</u>

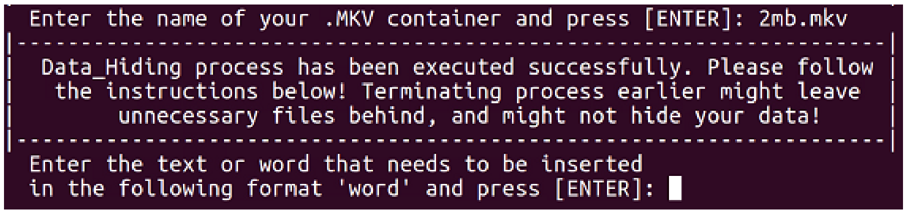
1. To hide the data in one of the selected MKV containers (provided, or can be used any downloaded from the internet), type: **./hide** inside the Terminal



```
maxim@maxim-XPS-15-9560:~/Desktop/final_fyp$ ./hide
|-----------------------------------------------------------------|
|       This script will allow you to hide the data inside your   |
|   .MKV container. Please follow the instructions below to launch |
|              data_hiding process successfully.                   |
|-----------------------------------------------------------------|
  Enter the name of your .MKV container and press [ENTER]:
```

   a. The above prompt will appear saying the process has begun

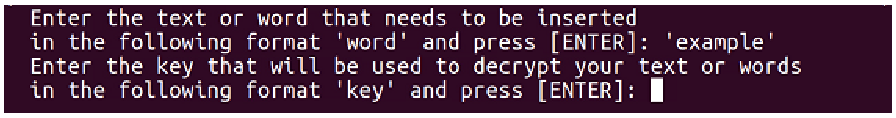
2. Enter the name of the MKV container, for example: **2mb.mkv**



```
 Enter the name of your .MKV container and press [ENTER]: 2mb.mkv
|-----------------------------------------------------------------|
|  Data_Hiding process has been executed successfully. Please follow |
|   the instructions below! Terminating process earlier might leave  |
|       unnecessary files behind, and might not hide your data!      |
|-----------------------------------------------------------------|
 Enter the text or word that needs to be inserted
 in the following format 'word' and press [ENTER]:
```

   a. The prompt will appear that data hiding process has been started

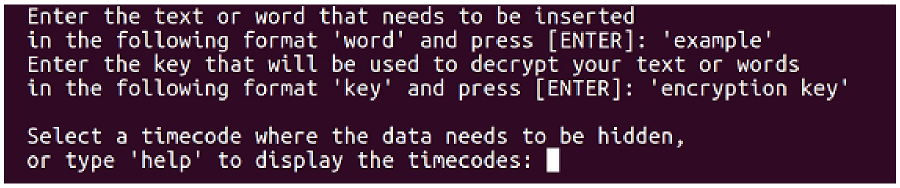
3. Enter the data that needs to be hidden in the following format, for example: **‘example’**



```
 Enter the text or word that needs to be inserted
 in the following format 'word' and press [ENTER]: 'example'
 Enter the key that will be used to decrypt your text or words
 in the following format 'key' and press [ENTER]:
```

   b. If the input is successful, the code will prompt the next step shown above

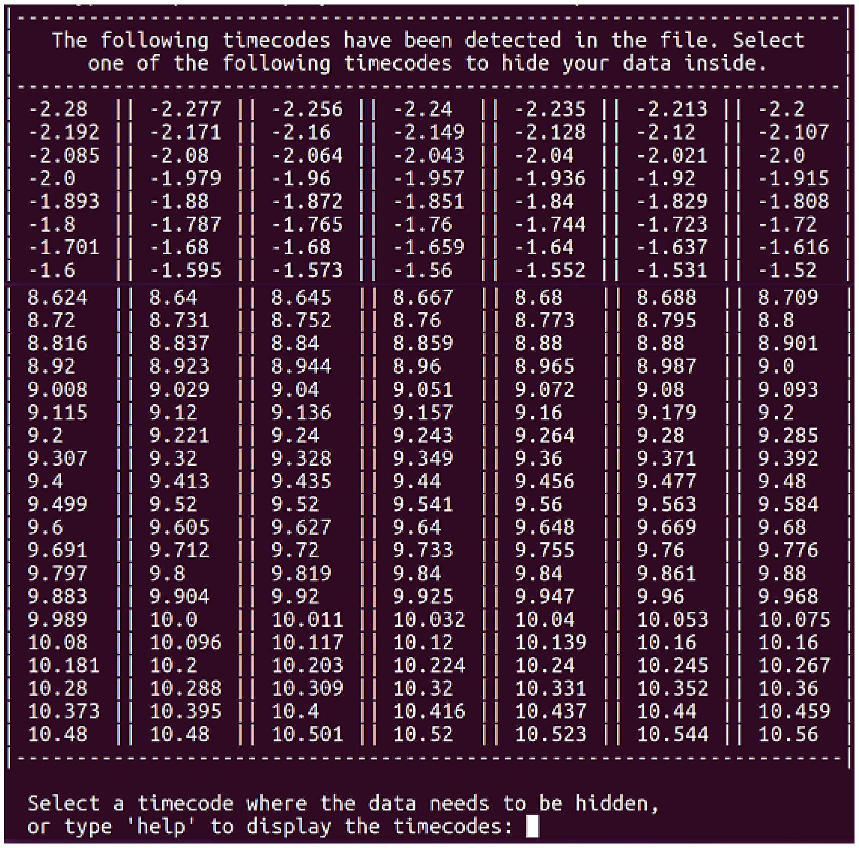
4. Enter the encryption key that will be used for decryption in the following format, for example: **‘encryption key’**



```
Enter the text or word that needs to be inserted
in the following format 'word' and press [ENTER]: 'example'
Enter the key that will be used to decrypt your text or words
in the following format 'key' and press [ENTER]: 'encryption key'

Select a timecode where the data needs to be hidden,
or type 'help' to display the timecodes: 
```

c. If the input of the key is successful, the code will prompt the next step shown above

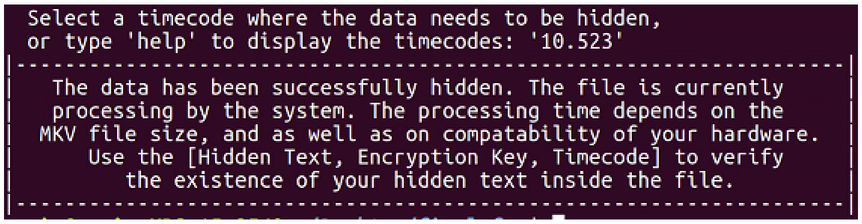
5. Enter ‘help’ to detect available timecodes (since the timecodes are not known), of course if timecode is known, it can be entered directly. For example, **‘help’**



```
|-----------------------------------------------------------------|
|   The following timecodes have been detected in the file. Select |
|      one of the following timecodes to hide your data inside.    |
|-----------------------------------------------------------------|
| -2.28  || -2.277 || -2.256 || -2.24  || -2.235 || -2.213 || -2.2   |
| -2.192 || -2.171 || -2.16  || -2.149 || -2.128 || -2.12  || -2.107 |
| -2.085 || -2.08  || -2.064 || -2.043 || -2.04  || -2.021 || -2.0   |
| -2.0   || -1.979 || -1.96  || -1.957 || -1.936 || -1.92  || -1.915 |
| -1.893 || -1.88  || -1.872 || -1.851 || -1.84  || -1.829 || -1.808 |
| -1.8   || -1.787 || -1.765 || -1.76  || -1.744 || -1.723 || -1.72  |
| -1.701 || -1.68  || -1.68  || -1.659 || -1.64  || -1.637 || -1.616 |
| -1.6   || -1.595 || -1.573 || -1.56  || -1.552 || -1.531 || -1.52  |
| 8.624  || 8.64   || 8.645  || 8.667  || 8.68   || 8.688  || 8.709  |
| 8.72   || 8.731  || 8.752  || 8.76   || 8.773  || 8.795  || 8.8    |
| 8.816  || 8.837  || 8.84   || 8.859  || 8.88   || 8.88   || 8.901  |
| 8.92   || 8.923  || 8.944  || 8.96   || 8.965  || 8.987  || 9.0    |
| 9.008  || 9.029  || 9.04   || 9.051  || 9.072  || 9.08   || 9.093  |
| 9.115  || 9.12   || 9.136  || 9.157  || 9.16   || 9.179  || 9.2    |
| 9.2    || 9.221  || 9.24   || 9.243  || 9.264  || 9.28   || 9.285  |
| 9.307  || 9.32   || 9.328  || 9.349  || 9.36   || 9.371  || 9.392  |
| 9.4    || 9.413  || 9.435  || 9.44   || 9.456  || 9.477  || 9.48   |
| 9.499  || 9.52   || 9.52   || 9.541  || 9.56   || 9.563  || 9.584  |
| 9.6    || 9.605  || 9.627  || 9.64   || 9.648  || 9.669  || 9.68   |
| 9.691  || 9.712  || 9.72   || 9.733  || 9.755  || 9.76   || 9.776  |
| 9.797  || 9.8    || 9.819  || 9.84   || 9.84   || 9.861  || 9.88   |
| 9.883  || 9.904  || 9.92   || 9.925  || 9.947  || 9.96   || 9.968  |
| 9.989  || 10.0   || 10.011 || 10.032 || 10.04  || 10.053 || 10.075 |
| 10.08  || 10.096 || 10.117 || 10.12  || 10.139 || 10.16  || 10.16  |
| 10.181 || 10.2   || 10.203 || 10.224 || 10.24  || 10.245 || 10.267 |
| 10.28  || 10.288 || 10.309 || 10.32  || 10.331 || 10.352 || 10.36  |
| 10.373 || 10.395 || 10.4   || 10.416 || 10.437 || 10.44  || 10.459 |
| 10.48  || 10.48  || 10.501 || 10.52  || 10.523 || 10.544 || 10.56  |
|-----------------------------------------------------------------|

Select a timecode where the data needs to be hidden,
or type 'help' to display the timecodes: 
```

d. If ‘help’ is entered, the above timecodes will appear (this is just a screenshot, it might be different depending on the file). And the next step will appear asking to select the timecode again

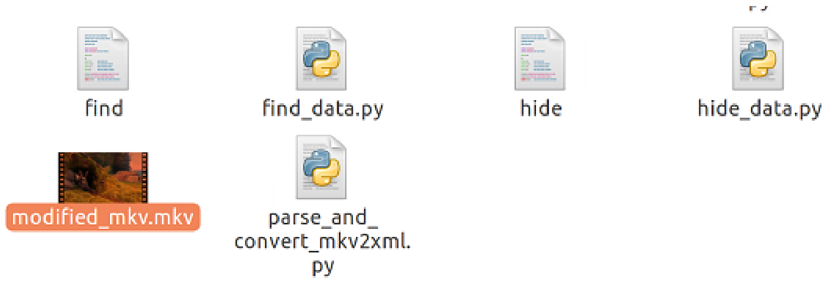
6. Select the timecode in the following format, for example: **’10.523’**



```
Select a timecode where the data needs to be hidden,
or type 'help' to display the timecodes: '10.523'
|-----------------------------------------------------------------|
|   The data has been successfully hidden. The file is currently   |
|   processing by the system. The processing time depends on the   |
|  MKV file size, and as well as on compatability of your hardware. |
|      Use the [Hidden Text, Encryption Key, Timecode] to verify    |
|         the existence of your hidden text inside the file.        |
|-----------------------------------------------------------------|
```

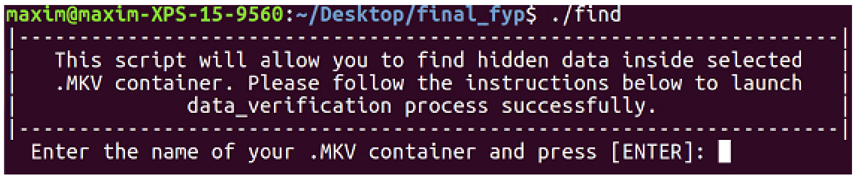
e. If timecode has been successfully detected, the above prompt needs to appear meaning that the data has been hidden.

7. If the data has been hidden, the file called **modified_mkv.mkv** needs to appear in your folder. This is the file with the hidden data inside.



Data Detecting Process

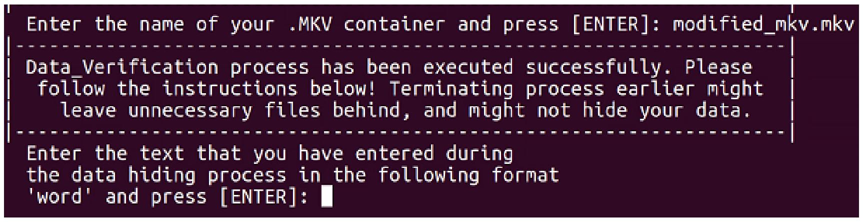
1. To detect the data in one of the selected MKV containers (from the data hiding process file name is **modified_mkv.mkv**), type: **./find** inside the Terminal



```
maxim@maxim-XPS-15-9560:~/Desktop/final_fyp$ ./find
|------------------------------------------------------------------|
|   This script will allow you to find hidden data inside selected |
|   .MKV container. Please follow the instructions below to launch |
|              data_verification process successfully.             |
|------------------------------------------------------------------|
 Enter the name of your .MKV container and press [ENTER]:
```

a. The above prompt will appear saying the process has begun

2. Enter the file name (make sure the file is in the folder, and the file has been created by running ./hide (data hiding)), for example: **modified_mkv.mkv**



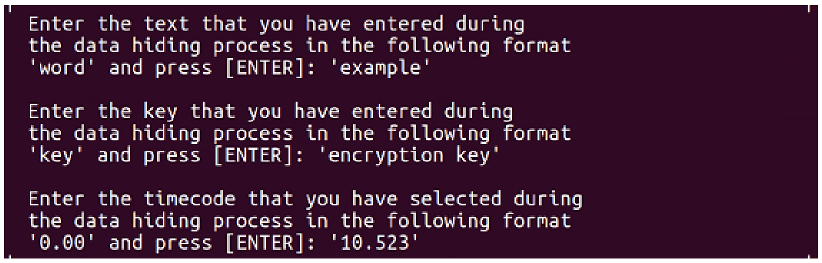
```
 Enter the name of your .MKV container and press [ENTER]: modified_mkv.mkv
|------------------------------------------------------------|
| Data_Verification process has been executed successfully. Please |
|  follow the instructions below! Terminating process earlier might |
|    leave unnecessary files behind, and might not hide your data.  |
|------------------------------------------------------------|
 Enter the text that you have entered during
 the data hiding process in the following format
 'word' and press [ENTER]:
```

a. The above prompt will appear saying that data detection algorithm is running

3. Type exactly the same text as been entered during the data hiding process, for example: **‘example’**

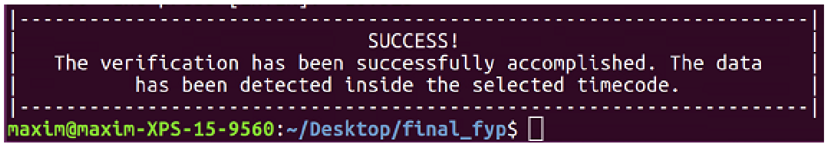
a. After that type the encryption key that has been used previously, for example **‘encryption key’**

b. After that type the timecode that has been selected during the data hiding process, for example **’10.523’**



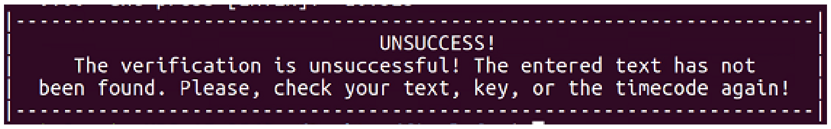
```
Enter the text that you have entered during
the data hiding process in the following format
'word' and press [ENTER]: 'example'

Enter the key that you have entered during
the data hiding process in the following format
'key' and press [ENTER]: 'encryption key'

Enter the timecode that you have selected during
the data hiding process in the following format
'0.00' and press [ENTER]: '10.523'
```

4. If all the data above has **been entered correctly** (identical to data hiding process), the following output needs to appear



```
|-------------------------------------------------------------------|
|                            SUCCESS!                               |
|   The verification has been successfully accomplished. The data   |
|            has been detected inside the selected timecode.        |
|-------------------------------------------------------------------|
maxim@maxim-XPS-15-9560:~/Desktop/final_fyp$
```

5. If some of the data is wrong, the following output needs to appear



```
|-------------------------------------------------------------------|
|                            UNSUCCESS!                             |
|     The verification is unsuccessful! The entered text has not    |
|  been found. Please, check your text, key, or the timecode again! |
|-------------------------------------------------------------------|
```

a. Error does not indicate where the issue is coming from to ensure that unnotarized users unable to identify the issue

Source Code

Ubuntu script: find

```
#!/bin/bash
# run from terminal ./find

echo "This script will allow you to find hidden data inside selected .MKV container. Please
follow the instructions below to launch data_verification process successfully."

read -p "  Enter the name of your .MKV container and press [ENTER]: " filename

if [ "${filename: -4}" == ".mkv" ]; then
        cat ${filename} | python parse_and_convert_mkv2xml.py > .tmp.xml &&
        python find_data.py .tmp.xml &&
        rm .tmp.xml
else
        echo "The file that you are trying to select is not .MKV container. Relaunch the
script with .MKV extension."
fi
```

Ubuntu script: hide

```
#!/bin/bash
# run from terminal ./hide

echo "This script will allow you to hide the data inside your .MKV container. Please follow
the instructions below to launch data_hiding process successfully.”

read -p "  Enter the name of your .MKV container and press [ENTER]: " filename

if [ "${filename: -4}" == ".mkv" ]; then
        cat ${filename} | python parse_and_convert_mkv2xml.py > .tmp.xml &&
        python hide_data.py .tmp.xml &&
        cat .hidden_data.xml | python convert_xml2mkv.py > modified_mkv.mkv &&
        rm .tmp.xml && rm .hidden_data.xml
else
        echo "The file that you are trying to select is not .MKV container. Relaunch the
script with .MKV extension.”
fi
```

parse_and_convert_mkv2xml.py (Shukela, 2012) – Access from Google Drive

convert_xml2mkv.py (Shukela, 2012) – Access from Google Drive

hide_data.py (contributed)

```
import sys
print('Data_Hiding process has been executed successfully. Please follow the instructions below! Terminating process earlier might leave unnecessary files behind, and might not hide your data!')

def text2ascii(a_word):
    ascii_store = ''
    for i in range(len(a_word)):
        ascii_ = str(ord(a_word[i]))
        if i != len(a_word): #add spaces to ascii's
            ascii_store += ascii_ + ' '
        else:
            ascii_store += ascii_
    return ascii_store

def pumpKey(a_word,k):
    new_k = ''
    count = 0
    for i in range(len(a_word)):
        if count+1 == len(k):
            new_k += str(k[count])
            count = 0
        else:
            new_k += str(k[count])
            count += 1
    k = new_k
    return k

def encr_ascii(ascii_x, ascii_k, x):
    encrypt_x = ''
    count = 0
    for i in range(len(ascii_x)):
        k_val = int(i % len(x)) # i mod (len X)
        count += 1
        if count > len(ascii_k): # if out of range, restarts the k_val
            k_val = len(ascii_x) - k_val
            count = 0
        k_mod = int(ascii_k[k_val])
        encrypt = int(ascii_x[i]) + k_mod + 33
        encrypt_x += str(chr(encrypt))
    return encrypt_x
```

```
def convert2hex(store_a_word):
    hex_word = ''
    hex_list = []
    for i in range(len(store_a_word)):
        count = 0
        for letter in store_a_word[i]:
            count += 1
            hex_word += hex(ord(letter))[2:]
            if count == len(store_a_word[i]):
                count = 0
                hex_list.append(hex_word)
                hex_word = ''
    combined_hex = ''
    for i in range(len(hex_list)):
        combined_hex += hex_list[i]
    return combined_hex

def get_timecodes(file_timecode):
    with open(file_timecode) as file_timecode:
        all_timecodes = []
        for line in file_timecode:
            if '<timecode>' in line:
                word = ''
                for x in range(len(line)):
                    if (line[x] not in '<timecode>') and (line[x] not in '</timecode>'):
                        word += line[x].rstrip() #combines the symbols/digits with the timecode
                all_timecodes.append(word) #stores the timecode in the table
        return all_timecodes

def return_timecodes(all_timecodes):
    print('The following timecodes have been detected in the file. Select one of the following
timecodes to hide your data inside.')
    max_size = 0
    for i in range(len(all_timecodes)):
            if len(all_timecodes[i]) > max_size:
                max_size = len(all_timecodes[i])

    word = ''
    count = 0 # used to create the columns (in this case 7)
    for i in range(len(all_timecodes)):
        while len(all_timecodes[i]) != max_size:
            all_timecodes[i] += ' '
        word += '| ' + all_timecodes[i] + ' |'
        count += 1
        if count == 7:
            print(word)
            word = ''
            count = 0
```

```
def hide_data(file, timecode, hexa_word):
    with open(file) as f_old, open(".hidden_data.xml", "w") as f_new:
        accessed_timecode = False
        for line in f_old:
            f_new.write(line)
            if (timecode in line) and (len(line) == 26+len(timecode)):
                accessed_timecode = True
            if (accessed_timecode == True) and ('<data>' in line):
                for i in range(len(hexa_word)):
                    f_new.write(hexa_word[i])
                accessed_timecode = False
    print('The data has been successfully hidden. The file is currently processing by the system. The processing time depends on the file size, and as well as on compatability of your hardware. Use the [Hidden Text, Encryption Key, Timecode] to verify the existence of your hidden text inside the file.')

def main(file):
    a_word = input('  Enter the text and press [ENTER]: ')
    k = input('  Enter the key and press [ENTER]: ')
    k = str(k)
    if len(k) < len(a_word):
        k = pumpKey(a_word,k)
    ascii_a_word = text2ascii(a_word)
    ascii_k = text2ascii(k)
    ascii_a_word = convert2hex(ascii_a_word)
    ascii_k = convert2hex(ascii_k)
    if len(ascii_k) < len(ascii_a_word):
        ascii_k = pumpKey(ascii_a_word,ascii_k)
    encrypt_a_word = encr_ascii(ascii_a_word, ascii_k, a_word)
    hex_list = convert2hex(encrypt_a_word)
    hexa_word = ''
    for i in range(len(hex_list)):
        hexa_word += hex_list[i] #gets pure hexadecimal word

    while True:
        all_timecodes = get_timecodes(file)
        select_timecode = input('\n  Select a timecode: ')

        if select_timecode not in all_timecodes and select_timecode != 'help':
            print('Selected time code is not found, try again!')
        elif select_timecode == 'help':
            return_timecodes(all_timecodes)
        else:
            hide_data(file, select_timecode, hexa_word)
            break

file = sys.argv[1]
main(file)
```

find_data.py (contributed)

```
import sys
print('Data_Verification process has been executed successfully. Please follow the instructions below! Terminating process earlier might leave unnecessary files behind, and might not hide your data.')

def text2ascii(a_word):
    ascii_store = ''
    for i in range(len(a_word)):
        ascii_ = str(ord(a_word[i]))
        if i != len(a_word):
            ascii_store += ascii_ + ' '
        else:
            ascii_store += ascii_
    return ascii_store

def pumpKey(a_word,k):
    new_k = ''
    count = 0
    for i in range(len(a_word)):
        if count+1 == len(k):
            new_k += str(k[count])
            count = 0
        else:
            new_k += str(k[count])
            count += 1
    k = new_k
    return k

def encr_ascii(ascii_x, ascii_k, x):
    encrypt_x = ''
    count = 0
    for i in range(len(ascii_x)):
        k_val = int(i % len(x))
        count += 1
        if count > len(ascii_k):
            k_val = len(ascii_x) - k_val
            count = 0
        k_mod = int(ascii_k[k_val])
        encrypt = int(ascii_x[i]) + k_mod + 33
        encrypt_x += str(chr(encrypt))
    return encrypt_x

def convert2hex(store_a_word):
    hex_word = ''
    hex_list = []
    for i in range(len(store_a_word)):
        count = 0
        for letter in store_a_word[i]:
            count += 1
            hex_word += hex(ord(letter))[2:]
            if count == len(store_a_word[i]):
                count = 0
                hex_list.append(hex_word)
                hex_word = ''
    combined_hex = ''
    for i in range(len(hex_list)):
        combined_hex += hex_list[i]
    return combined_hex

def generate_hexa_table(hexa_word):
    hexa_word_table = []
    count_max_word = 0
    count_max_length = 0
    word = ''
    for i in range(len(hexa_word)+1):
            if count_max_word <= 63:
                word += hexa_word[count_max_length]
                if (count_max_length + 1) == len(hexa_word):
            hexa_word_table.append(word)
                        break
                count_max_word +=1
                count_max_length += 1
            else:
                hexa_word_table.append(word)
                count_max_word = 0 #resets max length
                word = ''
    return hexa_word_table

def find_data(file, timecode, hexa_word):
    hexa_table = generate_hexa_table(hexa_word)
    count_hex = 0
    with open(file) as find_data:
        accessed_timecode = False
        accessed_data = False #data is not found
            data_found = True
            data_found_1 = 'no'
        for line in find_data:
            if (timecode in line) and (len(line) == 26+len(timecode)):
                        accessed_timecode = True
            if (accessed_data == True and accessed_timecode == True and data_found == True):
                        word = ''
                        for i in range(6,len(line)):
                            word += line[i]
                        if hexa_table[count_hex] in word:
```

```
                        count_hex += 1
                        data_found_1 = 'found'
                        if data_found_1 == 'found' and len(hexa_table) == count_hex:
                            return 'The verification has been successfully accomplished. The data
has been detected inside the selected timecode.'
                    else:
                    data_found_1 = 'not_found'
                        data_found = False
                        if data_found_1 == 'not_found':
                    return 'UNSUCCESS! The verification is unsuccessful! The entered text has
not been found. Please, check your text, key, or the timecode again! '
                if (accessed_timecode == True) and ('<data>' in line):
                    accessed_data = True
    return 'UNSUCCESS! The verification is unsuccessful! The entered text has not been
found. Please, check your text, key, or the timecode again! '

def main(file):
    a_word = input(' Enter the text that you have entered during the data hiding process:')
    k = input('\n  Enter the key that you have entered during the data hiding process: ')
    select_timecode = input('\n  Enter the timecode that you have selected during the data
hiding process: ')
    if len(k) < len(a_word):
        k = pumpKey(a_word,k)
    ascii_a_word = text2ascii(a_word)
    ascii_k = text2ascii(k)
    ascii_a_word = convert2hex(ascii_a_word)
    ascii_k = convert2hex(ascii_k)
    if len(ascii_k) < len(ascii_a_word):
        ascii_k = pumpKey(ascii_a_word,ascii_k)
    encrypt_a_word = encr_ascii(ascii_a_word, ascii_k, a_word)
    hex_list = convert2hex(encrypt_a_word)
    hexa_word = ''
    for i in range(len(hex_list)):
        hexa_word += hex_list[i]
    print(find_data(file, select_timecode, hexa_word))

file = sys.argv[1] #runs hidden file
main(file)
```