MP4 Steganography: Analyzing and Detecting TCSteg

Anthony Ramirez
Illinois Institute of Technology
201 East Loop Road
Wheaton, Illinois 60189-8489
630.682.6010
aramire2@hawk.iit.edu

Alfredo Fernandez
Illinois Institute of Technology
201 East Loop Road
Wheaton, Illinois 60189-8489
630.682.6010
aferna10@hawk.iit.edu

ABSTRACT

The MP4 files has become to most used video media file available, and will mostly likely remain at the top for some time to come. This makes MP4 files an interesting candidate for steganography. With its size and structure, it offers a challenge to steganography developers. While some attempts have been made to create a truly covert file, few are as successful as Martin Fiedler's TCSteg. TCSteg allows users to hide a TrueCrypt hidden volume in an MP4 file. The structure of the file makes it difficult to identify that a volume exists. In our analysis of TCSteg, we will show how Fielder's code works and how we may be able to detect the existence of steganography. We will then implement these methods in hope that other steganography analysis can use them to determine if an MP4 file is a carrier file. Finally, we will address the future of MP4 steganography.

Keywords

VeraCrypt; MP4; Steganography; Steganalysis; Detection; Hidden; Volume; Containers; Overt; Covert

1. INTRODUCTION

The MPEG-4 file type has become one of the most popular video file formats because its size, format, and playback support from most media players. It is the recommended file format for YouTube [1]. Its versatility has made it the most common video format found on the internet [2], surpassing other formats like AVI, FLV, and MOV. MP4 has universal support from mobile devices and web browsers.

MP4's universal support makes it a perfect candidate for use as a steganography carrier file. Its popularity makes it an unsuspecting file. Its file size can vary larger than other common carrier files, making it suitable for more covert content. In addition, the MP4 file format is very unrestrictive, making it perfect for development of new stenographic tools.

With this is in mind, MP4 does still over plenty of challenges. The media data in an MP4 container varies from file to file, as there exist many codecs used for audio and video, both lossless and lossy. In addition, the file format's unrestrictive nature means the container does not adhere to a standard format, which can challenging.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
RIIT'16, September 28–October 1, 2016, Boston, MA, USA.
© 2016 ACM. ISBN 978-1-4503-4453-1/16/09...$15.00.
DOI: http://dx.doi.org /10.1145/2978178.2978181

While some tools have been created to hide data in MP4 files, most of these tools of end of file insertion techniques, making them an easy target for detection. For our purpose we wanted to find a tool that not only made it difficult detect the existence of covert data, but also would leave the file seemingly unaltered when played using a media player. What we discovered was Martin Fiedler's TCSteg. TCSteg allows the user to inserts a TrueCrypt hidden volume into any MP4 file. If used correctly, TCSteg is very difficult to detect. TCSteg uses the full functionality and plausible deniability of TrueCrypt, making it a simple method to hide encrypted data in an MP4 that can be played with any media player after modification has been completed.

Our goal is to show that with analysis, it is possible to detect the presence of steganography in a TCSteg MP4 file. Using several techniques for analysis, we will show that there are some consistence details that may indicate the existence of a TrueCrypt hidden volume. We hope that future steganography analysis will use these techniques to analyze MP4 files, as it is more than likely that future methods of hiding data will be detectable using the same methodologies we followed.

2. STEGANOGRAPHY

Steganography is the act of concealing the existence of some data. This can be seen as an alternative to cryptography, where the goal is to conceal the meaning of data. Steganography can be as simple as invisible ink on paper, or as complex as digital images with hidden data embedded in them. In both of these scenarios, we have information that we want to remain hidden, our covert message, a medium that appears innocent, our overt file, and a method used to hide the existence of the covert message into the overt file. There are three methods that can be used to hide data: generation, substitution, and insertion.

2.1 Generation Method

Generation method uses the covert file to create an overt file. An example of this would be a sentence where every other word was a part of the covert message. In the case of an MP4 file, this would require the creation of a video by which the viewer would be able to uncover a hidden message from viewing

2.2 Substitution Method

Substitution method requires replacing some of the data from the overt file with the covert file. This is done through the use of the least significate bits. We can modify the least significate bits of an overt file without modifying its appearance. This is a common method for image steganography.

2.3 Insertion Method

Insertion method inserts the covert data in a location that will not affect how the overt data is processed. An example of this would be inserting covert data at the end of the file of an overt file. For instance, if a file format has a field that is unused by the file processor, this field would be perfect for insertion. For MP4 steganography, this is the most common method used to insert covert data.

3. MPEG-4

The MP4 file format was developed by the Moving Pictures Experts Group and is derived from the QuickTime file structure developed by Apple. This file format has become one of the most popular video formats because of how well it compresses. An MP4 file maintains great quality at a relatively small size which allows online video sharing to become faster and more efficient. The ability to store video, music, images and text allows for advanced context such as 3D graphics and menus to incorporate as part of the file. In general, video formats are considered to be good carrier files due to their larger size which allows more potential to hide data without raising suspicion based on size.

3.1 Main atoms

The structure of an MP4 file consist of data units called atoms. Atoms work in a hierarchy where sub-atoms can be contained within an atom for organizational purposes. There are three main atoms required for an MP4 file and are abbreviated as 'ftyp', 'moov', and 'mdat'. The only requirement for these atoms is that 'ftyp' must come before 'moov'or 'mdat'. The position of 'moov' or 'mdat' can be interchanged.

3.1.1 FTYP

The 'ftyp' main atom is the main file header for the MP4 file. This atom identifies the file type in order to determine compatibility. As the main file header, 'ftyp' must be the first atom. A video with audio will contain a track for each. A third track is seen when text, for example subtitles, are added to the video.

3.1.2 MOOV

The 'moov' main atom is important because it contains metadata used to provide instructions on how to handle the media. This atom is heavily nested with inner sub-atoms to determine number and type of tracks, location of sample data, and more.

3.1.2.1 STCO

The 'stco' sub-atom contains the chunk offset table and is located inside the 'moov' atom. This atom allows the mapping of samples as a way to sequentially reference the location in the 'media data. The chunk-offset 'stco' references 32-bit offsets but there is a variant sub-atom 'co64' that can reference 64-bit.

3.1.3 MDAT

The 'mdat' main atom takes up the majority of the file size as it is where media data for video frames, audio, and images are stored.

4. TrueCrypt and VeraCrypt

TrueCrypt and VeraCrypt are open source tools that provide the ability to perform on the fly encryption. VeraCrypt is a fork continuation of TrueCrypt since TrueCrypt was discontinued. These tools can create logical encrypted volumes which can contain any type of data. In order to mount a volume and have access to the contents, a password must be provided to decrypt the data [7].

4.1 Inner/Outer Volume Structure

4.1.1 The Outer Volume

VeraCrypt and TrueCrypt have the ability to create two types of volumes. The first is the outer volume which is just the allocation of a certain amount space. The header of the outer volume starts at byte offset 0 and can be up to 65,535 bytes.

4.1.2 Hidden Volume/Inner volume

The second type of volume is a hidden volume which is contained within the outer volume. The header of the hidden volume is located right after the outer volume and ranges from 65,536 to 131,072 bytes.

4.1.2.1 Steganography features

Interestingly, both of these tools encrypt the entire contents of the volume, even the header, with randomly encrypted data. This offers plausible deniability because there is no identifiable volume or signature to be able to point at and attempt to prove that there is anything other than random data. TrueCrypt and VeraCrypt is able to find the headers since they are always located at either byte offset 0 for outer volumes or byte offset 65,536 for hidden volumes.

5. TCSteg

TCSteg is a python script that was developed by Martin Fiedler and later updated by Vladimir Ivanov [5]. The script essentially performs a VeraCrypt and MP4 merge. To work, the VeraCrypt container must have an outer and inner hidden volume. TCSteg then rearranges the atoms in the MP4 file to ensure that 'mdat' comes before 'moov'. After, the outer volume is stripped off and the hidden volume is merged into the MP4 at the start of the 65,536 offset byte. The 'mdat' is then moved to the end of the VeraCrypt volume and a new 'mdat' that spans the volume and the 'mdat' is created. Fake 'mdat' data that resembles the real 'mdat' is placed before the volume header. Last, the chunk offset tables are changed to adjust to the new locations of the media samples. Figure 1a, shows an example of the merge looks.

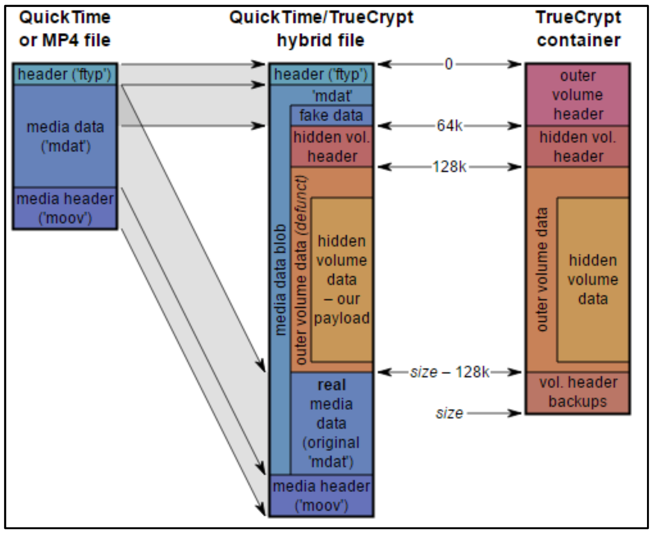


Figure 1. Shows the merge between a volume and an MP4

6. Analysis

Our goal for our analysis is to be able examine any MP4 file, and say to some degree of certainty that specific traits of a file indicate the use of TCSteg.

While it is almost impossible to say with one hundred percent certainty that any file is a true positive or negative for steganography, we hope to shed some light on techniques that could help in future analysis.

For our analysis of TCSteg, we had to consider the type of steganography being used by this tool. In this case, TCSteg uses an insertion technique to hide data within the MP4 carrier file. As stated in section 5, we know the constraints that TCSteg must follow for the embedding process to take place. In addition to access to the TCSteg source code, we also have access to the overt, covert, and steganography file [6].

6.1 Tools for Analysis

Besides the use of VeraCrypt, Python, and Fiedler/Ivanov's TCSteg, we used several additional tools for analysis. There are not many tools that have been designed for MP4 steganography analysis. This forced us to improvise on the tools we could use. In addition to the already listed tools and a hex editor, WinHex, we used MediaInfo and ISO Viewer.

6.1.1 MediaInfo

MediaInfo is an open source application that displays various metadata about a video file [3]. MediaInfo supports most video formats, and has a number of display options.

6.1.2 Iso Viewer

Iso Viewer was created by Sebastian Annies [4]. ISO Viewer is a Java application that allows MP4 files, along with other ISO 14496 formats to be parsed. Iso Viewer gives a verbose view of the MP4 atoms, including offsets, flag details, and a hex viewer.

6.2 File Structure Analysis

In our goal to detect if a MP4 file has be altered by TCSteg, there are some assumptions we can make. TCSteg modifies the structure of an MP4 file in several ways.

6.2.1 Atom Order

An MP4 file modified by TCSteg will always have an atom order of ftyp, followed by mdat, followed by moov. The reason for this, as stated before is to accommodate hiding volume. This structure is necessary for mounting of said hidden volume, and will not mount if structured differently.

6.2.2 mdat Analysis

In addition to the mdat being second in the atom hierarchy, we also need to ensure that there is data located in the 64 kilobyte to 168 kilobyte ranges of the file. A TCSteg MP4 file will always have seemingly random looking data in this range. This is because the hidden volume header will always be located in the range 64 kilobytes to 128 kilobytes. In addition, we know that the smallest possible hidden volume is 40 kilobytes. If there is no data in that range, or padding in the file at this range, we can assumethe file is not a TCSteg MP4.

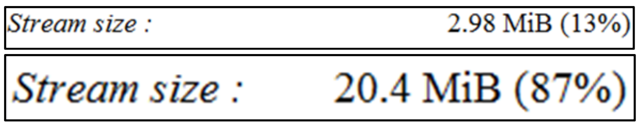
6.3 Methods of Detection

Fiedler recommends four methods to detect TCSteg MP4 files. The first he recommends is to analyze the bitrates of the file in relation to the actual file size. "The only other ways would be (1) a sophisticated packet-by-packet analysis of the mdat data that would find out that from offset 65,536 on, there's not only random-looking compressed data, but random-looking garbage; (2) checking for a repetition of the first 64k of mdat somewhere later in the file and (3) seeing that there's much unused space in the mdat that isn't referenced by any chunk offset table." [5]

6.3.1 Bitrate Analysis

The bitrate of the MP4 file will give us an indication of what we should expect the size of the file to be. For example, if we have a file where both the audio and video streams of the file have a combined bitrate of 1 megabyte per second, a length of 10 seconds, and a file size of 10 megabytes, we could say that the file appears to be in order. If we had a file of the same attributes, but was 20 megabytes large, we could say that the file has 10 megabytes unaccounted for.

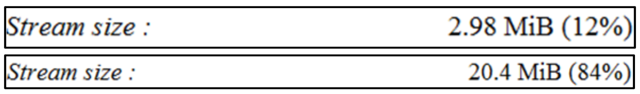
For this analysis we used the tool MediaInfo. MediaInfo gives an accurate measurement of the bitrates of the audio and video streams. In Figure 2, we see that the stream sizes for the audio and video of a clean MP4 file, along with their respective percentage of the total file:


Stream size : 2.98 MiB (13%)

Stream size : 20.4 MiB (87%)

Figure 2. Shows the Audio (top) and Video (bottom) stream sizes of a clean MP4 file using MediaInfo.

As we see in Figure 2, the percentages add up to 100, as they should. If we perform the same analysis on a TCSteg video, we will see results similar to what is shown in Figure 3:


Stream size : 2.98 MiB (12%)

Stream size : 20.4 MiB (84%)

Figure 3. Shows the Audio (top) and Video (bottom) stream sizes of a TCSteg MP4 file using MediaInfo.

Just as we saw in Figure 2, we see the same stream sizes in Figure 3, but the percentages are not the same. This is because there is content hidden in the mdat of this MP4 file.

While this method is a simple way to test for the existence of a hidden volume in a MP4 file, it is not accurate. While it would make sense for a clean MP4 file to always have stream sizes that add up to 100 perfect, this is not always true. In some of the files we have encountered, there has been data placed in the mdat that has no effect on either the video or audio data. It could be data that makes reference to the encoder, the software used to edit the file, or even just padding added to the file. In smaller video files, this degrades the accuracy of our stream size measurements.

To further complicate matters, if the MP4 file is large, bitrate analysis comes less accurate. If a user hides a smaller volume in a large MP4 file, the respective stream size percentages would not accurately represent the existence of that volume.

While bitrate analysis is a quick and simple method to analyze an MP4 file for TCSteg, it does not give an accurate assessment for all files.

6.3.2 Statistical Analysis

For our second method of detection, Fiedler recommends analysis determine if data is compressed or encrypted. What we know what encrypted data is that it should appear more random than compressed data. As stated earlier, an MP4 file's mdat will contain compressed or encoded data. If that MP4 file is using TCSteg, it will also contain encrypted data. We also know that if a TCSteg MP4 file did contain encrypted data, then that data would have to exist in the 64 kilobyte to 168 kilobyte range of said file. We came to the conclusion that there must be a way to perform a statistical analysis of this range.

6.3.2.1 Chi-squared

Chi-squared is a method of statistical analysis that allow a data sample to be compared to an expected result, and determine its relation to those expected results. In Figure 4, we see our chi-squared equation:

$$x^2 = \sum_{0}^{i} \frac{(O_i - E_i)^2}{E_i}$$

Figure 4. Shows the chi-squared equation.

In Figure 4, O represents the observed count of the value i, E represents the expected count of the value i, and x^2 represents our chi-squared value.

Let us consider an example of how we might use chi-squared. Say we have a 6-sided die, and want to prove that the die is not a "loaded" die. We know in a normal die, if we rolled it six times, we would expect that each side would land face-up one time. We also know that in reality, this almost never happens perfectly. We can use our expected results against a large sample die rolls, and show that there is a statistical significant enough of a difference that would indicate that the die is not uniformly random. In this example, our observed count would be the number of rolls we observe for each side, and our expected count would be the number of rolls we expected for each side.

6.3.2.2 Chi-Squared Analysis

In the case of encrypted data, we would expect for it to appear uniformly random. We also know that we will have a total 256 possible observable values. For each observable value, our expected count will be the same, based on our sample size. As we stated before, we analyzed the 64 kilobyte to 168 kilobyte range, giving us a total of 106496 observable bytes. For each possible observable value, we expect to see a count of 416.

Using a Python script we wrote, we were able to tally the count of each observed value and calculate the chi-squared value of our sample range. In addition to creating the script, we used Microsoft Excel to create a chi-squared distribution graph, so that our chi-squared values would have a corresponding P value, probability. Figure 5 shows that graph:

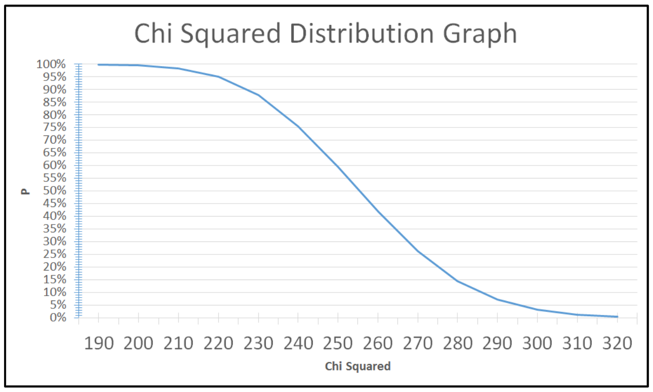
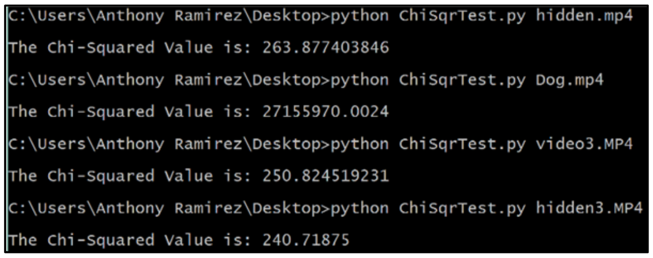


Figure 5. Shows our chi-squared distribution graph.

The results of our Python script can be seen in Figure 6:



```
C:\Users\Anthony Ramirez\Desktop>python ChiSqrTest.py hidden.mp4
The Chi-Squared Value is: 263.877403846
C:\Users\Anthony Ramirez\Desktop>python ChiSqrTest.py Dog.mp4
The Chi-Squared Value is: 27155970.0024
C:\Users\Anthony Ramirez\Desktop>python ChiSqrTest.py video3.MP4
The Chi-Squared Value is: 250.824519231
C:\Users\Anthony Ramirez\Desktop>python ChiSqrTest.py hidden3.MP4
The Chi-Squared Value is: 240.71875
```

Figure 6. Shows the calculated chi-squared values of four files

In Figure 6, we see four files that have been calculated. "hidden.mp4" and "hidden3.mp4" are both TCSteg MP4 files, while "Dog.mp4" and "video3.mp4" are both clean MP4 files. In our file "Dog.mp4" we see a large value for chi-squared. This is expect. That particular video had very little content in the sample range, causing it to have a high value resulting in a zero probability. In our other files, we see relatively low values for chi-squared, which results in a significant probability value. What should result in some confusion is the results of "video3.mp4". It is clean video, with a statistically significant chi-squared value. Why might this happen? We have determined that while the encrypted data appears statically random, encoded data can also appear random.

With our chi-squared analysis, we are not trying to prove that data is encrypted. We are attempting to show a null hypothesis. This means that we are trying to find data that does not match the statistical variation of encrypted data. While chi-squared does have value as a part of our analysis, it alone cannot provide proof of the existence of TCSteg.

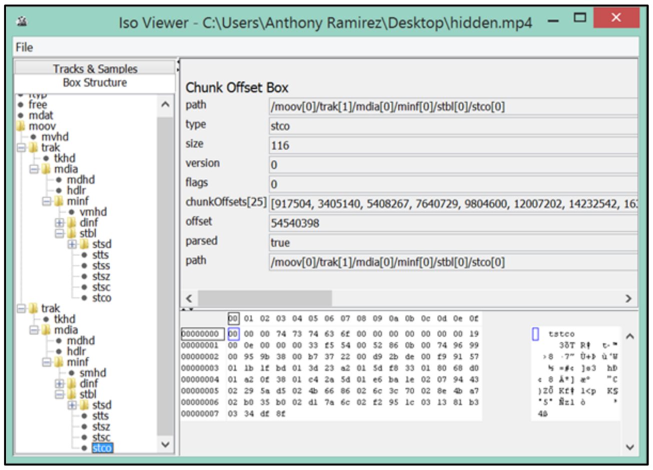
6.3.3 Search for Repetition

Fiedler's other recommendation for detecting TCSteg is to check for repetition the first 64 kilobytes of the mdat later in the file. We chose not to perform this analysis for several reasons. This method can be computationally long with any large files. We also discovered that the updated version of TCSteg, authored by Ivanov, removed duplicated encoding signatures, making this method fruitless. As the other methods seemed to be more effective when compared to this method, we chose to focus our resources on them.

6.3.4 Analysis of Chunk Offset Table

The last method for detection that Fiedler recommends is looking for unreferenced data in the mdat using the stco. As stated in section 3.1.2.1, the stco, also known as the chunk offset table, references the location of each media chunk in the mdat. If it is not reference by the stco, it is not part of the files media content.

We can use ISO Viewer to determine the location of the first chunk in the stco. In Figure 7 we see the parsing of our TCSteg MP4 file:



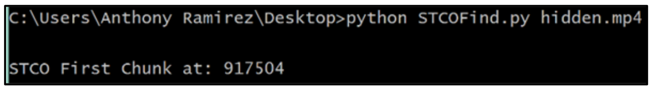
Chunk Offset Box

path	/moov[0]/trak[1]/mdia[0]/minf[0]/stbl[0]/stco[0]
type	stco
size	116
version	0
flags	0
chunkOffsets[25]	[917504, 3405140, 5408267, 7640729, 9804600, 12007202, 14232542, 16
offset	54540398
parsed	true
path	/moov[0]/trak[1]/mdia[0]/minf[0]/stbl[0]/stco[0]

```
         00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00000000 00 00 00 74 73 74 63 6f 00 00 00 00 00 00 00 19
00000001 00 0e 00 00 00 33 f5 54 00 52 86 0b 00 74 96 99
00000002 00 95 9b 38 00 b7 37 22 00 d9 2b de 00 f9 91 57
00000003 01 1b 1f bd 01 3d 23 a2 01 5d f8 33 01 80 68 d0
00000004 01 a2 0f 38 01 c4 2a 5d 01 e6 ba 1e 02 07 94 43
00000005 02 29 5a d5 02 4b 66 86 02 6c 3c 70 02 8e 4b a7
00000006 02 b0 35 b0 02 d1 7a 6c 02 f2 95 1c 03 13 81 b3
00000007 03 34 df 8f
```

Figure 7. Shows Iso Viewer display of stco data

We see that the first chunk is located at byte 917504. Iso Viewer allows us to parse through each atom of the file. While it is an effective tool, we opted to create a Python script that could automate this process. The Python script parses our MP4 file for the stco atom. We then determined the first chunk offset. Using that value we can determine where the first chunk is referenced in the mdat. Figure 8 shows the results from running that script:



```
C:\Users\Anthony Ramirez\Desktop>python STCOFind.py hidden.mp4

STCO First Chunk at: 917504
```

Figure 8. Shows the parsed location of stco first chunk.

For this file, we found that the first chunk is located at byte 917504 of the file, just as we saw it in Iso Viewer.

Once we have determined the location of the first chunks location, we can determine if the location would be in the range of the hidden volume. As stated before, we would see the hidden volume always existing in the byte range 65536 to 172032. In our "hidden.mp4", we see that the first chunk starts well out of the range. This is a strong indication that the MP4 file may have a hidden volume.

7. Conclusion

We have determined that while TCSteg is difficult to detect, using our method of detection we can make assessment on the existence of the hidden volume. We also have concluded that while there are not many tools for MP4 steganography analysis on the market, there are some applications that already exist that can be used for this purpose. We feel in the future it is likely other insertion techniques for MP4 steganography will be developed. As the MP4 file is still being explored, it will be interesting to see what other locations data can be hidden.

8. REFERENCES

[1] Recommended upload encoding settings - YouTube Help. *Support.google.com*, 2016. https://support.google.com/youtube/answer/1722171.

[2] Orlin, J. Survey: MP4 Is Top Format For Web and Mobile Videos. *TechCrunch*, 2012. http://techcrunch.com/2012/04/17/survey-mp4-is-top-format-for-web-and-mobile-videos/.

[3] MediaInfo. *Mediaarea.net*, 2016. https://mediaarea.net/en/MediaInfo.

[4] Annies, S. sannies/isoviewer. *GitHub*, 2016. https://github.com/sannies/isoviewer.

[5] Feidler, M. KeyJ's Blog : Blog Archive » Real Steganography with TrueCrypt. *Keyj.emphy.de*, 2011. http://keyj.emphy.de/real-steganography-with-truecrypt/.

[6] Cimarron Systems, LLC.,. *Elements of the H.264 Video/AAC Audio MP4 Movie*. Cimarron Systems, LLC., Evergreen, Colorado, 2014.

[7] VeraCrypt - Documentation. *VeraCrypt*, 2016. https://veracrypt.codeplex.com/documentation.