Steganography and Data Hiding in Flash Video (FLV)

Jason Paul Cruz, Nathaniel Joseph Libatique, and Gregory Tangonan

Electronics, Communications and Computer Engineering Department
and the Ateneo Innovation Center
Ateneo de Manila University, Quezon City, Philippines

***Abstract*—Due to increased security threats experienced today, confidential information, such as medical records and banking data, are at risk. Thus, multiple layers of data protection beyond conventional encryption must be considered. The current study presents the design and implementation of a steganographic protocol and a suite of tools that can automatically analyze a flash video (FLV) format and effectively hide information within it for application in a digital records environment. This study proposes several methods of hiding information within an FLV and discusses the corresponding advantages and disadvantages of such methods. The proposed steganographic methods are analyzed qualitatively by using auditory–visual perception tests and quantitatively by using video tags evolution graphs and histograms and RGB averaging analysis. This study assumes a system where sensitive data is embedded inside FLVs and then transmitted to several recipients who hold varying access authorization levels.**

Keywords—steganography; flash video; data hiding; file security; software

I. INTRODUCTION

Steganography is the art and science of communicating in a way that hides the existence of the communication. In contrast to cryptography, where anyone is allowed to detect, intercept, and modify messages without violating certain security premises guaranteed by a cryptosystem, steganography aims to hide confidential messages inside other messages, known alternatively as carriers, in a way that does not allow anyone apart from the sender and intended recipient to detect the presence of the secret data. The modified carrier, which includes the secret data, is referred to as a stego-object, or in this research's case a stegoFLV.

Due to increasing security threats brought about by modern malevolent technology, confidential information, such as medical data and banking data, are at risk. Such sensitive data is typically stored in a central location or database that is protected by several layers of security, such as the use of passwords and encryption. However, many applications require the sharing of sensitive data to multiple users outside the boundaries of centrally administered databases, essentially exposing the data out in the open. Multiple pieces of confidential information are packaged in cover files and then sent out, read, shared, processed, further passed on, and possibly eventually re-enter the secured central location/database or, even in their various forms, stay outside indefinitely. A user can access information by interacting with a secured central server, but outside this secured wall, a user can access information hidden inside a cover file by using password/s and a priori knowledge that data is indeed hidden inside the file, especially in the context where a user cannot remotely access a central server, as shown in Fig. 1.

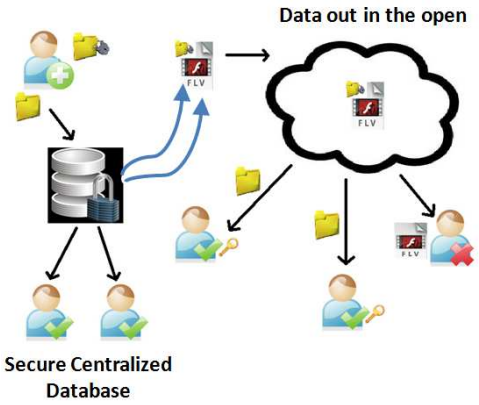

Figure 1. Encryption vs. Data Hiding: Accessing sensitive information in a secured centralized database vs. accessing and sharing it out in the open.

A data hiding protocol is necessary in the context of peer-to-peer sharing, multiple-level authorizations, and secret information processing. This study discusses the design and implementation of applications that can analyze a flash video (FLV), as the carrier, and apply effective methods for hiding secret data or stegofile within it. These applications include a high level of control for sharing a stegofile that may contain multiple pieces of information among multiple intended recipients, each with different levels of access, while looking like a normal and non-suspicious video to third parties or observers. This study presents a nonconventional means of providing sensitive data protection through steganographic techniques [1].

For example, a patient may secure his/her confidential medical data, such as patient information, diagnoses, past drug use, lab test results, and genetic predisposition to various diseases, among others, inside an FLV of his/her 2D echocardiogram by using the developed video steganography software, as shown in Fig. 2. The hiding process involves the compression of the secret data to a rar file and then securing it with password/s (the password protection is implemented using the DotNetZip library, which allows the compilation of files into a rar file with AES-encryption security) [2], [3]. The patient then sends this stegoFLV to a location, such as a personal website, where multiple recipients with different levels of access may retrieve the corresponding data associated with them provided they have the correct password/s to access the data (e.g., the head physician has access to all the files,

while the medical technician only has access to the lab results, or the nurse only has access to the lab results and past drug use, or the wife/husband only has access to the patient information, and so on).

Many steganographic techniques have been applied to different cover files. Some of the commonly used methods include the addition of spaces and tabs in document and text files, the least significant bit encoding and the frequency domain encoding through discrete cosine transform to hide secret data inside images, and the low-bit encoding and the use of discrete Fourier transform to hide data inside audio files [4]–[6].

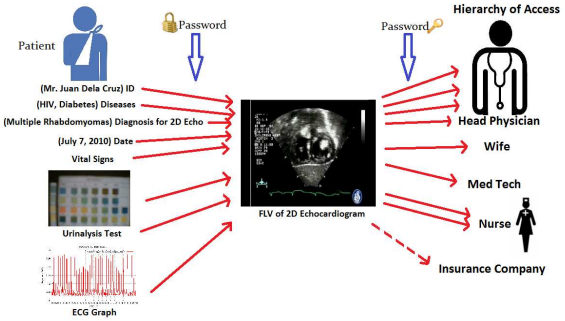

Figure 2. System diagram of sharing to multiple recipients with different levels of access.

However, only few studies on the application of steganography on videos, specifically FLVs, have been reported. A notable study is that of Mozo et al. [7], who discussed their analyses and experiments on the FLV file format and on the implementation of a C++ program that embeds and extracts data in an FLV and compresses an FLV. According to David Salomon, author of Data Privacy and Security, the main features of a data hiding algorithm are as follows [6]: (1) Embedding capacity (or payload): the amount of data that can be hidden in the cover; (2) Invisibility (or perceptual transparency): the amount of distortion to the cover which cannot be measured numerically but rather tied to an observer's visual or auditory perception; (3) Undetectability: the ability of an algorithm to maintain the statistical properties of the cover file; (4) Robustness: the measure of the algorithm to maintain the secret data even after changes to the stego-object; and (5) Tamper Resistance: the ability to make the alteration or deletion of the secret data difficult.

In the current study, the results of Mozo et al. were taken into consideration and were improved upon by designing and implementing a suite of tools for the analysis of the FLV file format and for the application of steganography on FLVs. Using these tools, different steganographic methods for hiding information inside an FLV were implemented, and the corresponding strengths and weaknesses of these methods guided by the features provided by Salomon were analyzed. The characteristics of the stegoFLVs were analyzed based on their corresponding video tags evolutions and histograms, and the results were compared with those of the carriers. In addition, the quality of the stegoFLVs was analyzed qualitatively by using auditory–visual perception tests and quantitatively by using the ROI_RGB_averaging software developed by Villegas [8], which generates the average RGB contents of a video. The development of the various steganographic methods and application software was inspired by the use of steganography against steganography based on the article "Spy vs Spy" [9]. The extraction feature of the video steganography program determines whether an FLV is "stegged" or contains hidden data.

II. FLV File Format

To effectively apply steganography on FLVs, the format and structure of an FLV file should first be investigated and fully understood. In this study, a hex editor [10] was used to investigate the actual contents of FLVs in the form of hexadecimal values. The FLV file format was chosen as the carrier because of the following reasons: (1) It is very popular and widely used, as seen in Youtube, Google Videos, and various personal websites; (2) It has audio and video, making the testing and experimentation easy and accurate; (3) Its typical file size is not too large (compared to the size of an avi or mp4 video) and not too small (compared to the size of an image), making it a great format to hide data without creating suspicion and making its distribution and retrieval easy; and (4) It has a simple file structure, and thus, its format is not difficult to understand.

The FLV file format is composed of a short header, followed by the metadata, and then interleaving audio and video tags or packets [11]. The FLV header is composed of the first three hex values of "46 4c 56", which translates to "FLV" in its hexadecimal string value, followed by the version, flags, and offset. After these values is a sequence of tags until the end of file (EOF). The tag types are composed of 0x08 for AUDIO, 0x09 for VIDEO, and 0x12 for META. Each tag contains the type, body length, timestamp, timestamp extended, streamID, and the body (actual data). After each tag is the previous tag size, which should always be equal to the size of the actual data in bytes plus 11 bytes corresponding to the tag type (1 byte), body size (3 bytes), timestamp (3 bytes), timestamp extended (1 byte), and stream Id values (3 bytes) of the tag.

III. Automated Analysis of an FLV

Prior to performing experiments on the FLV file format and applying various steganographic techniques, important parameters, such as the number, sizes, timestamps, and locations of the video and audio tags, should be known because they are the actual data that will be altered and modified. Such parameters are difficult to locate and record manually, especially if the FLV to be analyzed is large in size, because the number of bytes that will be analyzed when opened in a hex editor will be the total number of bytes of the FLV. For example, an FLV with a size of 1,500,000 bytes may take a long time to analyze manually and is prone to human errors. Therefore, a program developed in Visual Studio C# was designed and created to automate the analysis of the FLV file format. The automated FLV analysis software has the following features: (1) Determine if the chosen file is an FLV by analyzing the header; (2) Locate the position of each tag; (3)

Determine the type of each tag (whether it is a metadata, audio, or video tag); (4) Determine the size of each tag; (5) Determine the timestamp of each tag; and (6) Count the total number of metadata, audio, and video tags of an FLV. Fig. 3 shows a screenshot of the automated FLV analysis software.

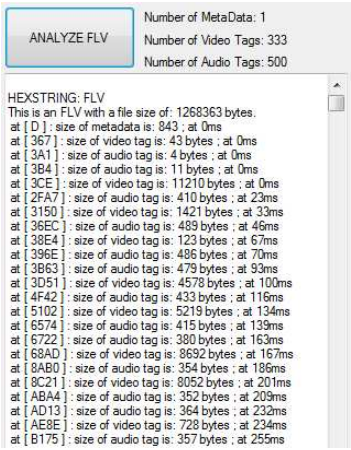

Figure 3. Program that automates the analysis of an FLV.

The automated FLV analysis software also generates Excel files that record some of the parameters (i.e., the type, sizes, and timestamps of the metadata, video, and audio tags) for further analysis. The creation of the automated FLV analysis software was a vital part in this study. Given the data obtained from the software, the experiments on the manipulation of the FLV file and on the application of different steganographic techniques were made possible and much easier.

IV. Experiments and Results on FLV Manipulation

A. FLV Header is Modified

When the FLV header was modified, wherein random values were added such that the header contained other values other than the accepted hex values, the modified FLV was corrupted and did not play at all. Thus, the FLV header should not be modified. Moreover, the modified FLV was also corrupted when random data was deleted across the FLV. The FLV will immediately stop playing at the timestamp of the deleted until the EOF. An FLV will not function properly if data are missing such that the value of the body size of a tag is not equal to the size of the actual data, if the previous tag size is not equal to the size of the actual data plus eleven, and if the timestamps are not in chronological order.

B. Tags are Removed in Whole

When video or audio tags were removed in whole, including the tag type, body size, timestamp, extended timestamp, streamId, the actual data, and the previous tag size, the modified FLV played the entire length but with distortions on the corresponding timestamp of the removed tags. Even though the FLV format did not contain conflicts, actual data was missing at a specific point in time throughout the duration of the video. Longer and more obvious distortions will be encountered if more tags are removed, especially if they are consecutive tags and timestamps. For example, when a video tag was removed somewhere in the 4-second mark of an FLV, the modified FLV contained some distortions compared with the carrier, as shown in the red circle in Fig. 4.

Figure 4. Screenshots of the playback of the original FLV (top) and modified FLV (bottom) with deleted tags.

When video tags were deleted at the start and middle of the FLV (somewhere in the 1-second and 4-second marks), both distortions in the respective timestamps were encountered. When a big chunk was removed (almost three-fourths of the entire size of the carrier), the modified FLV played normally at first, paused at the 3-second mark (no audio and video), and then continued playing after the 9-second mark until the EOF. These results show that the FLV plays in a chronological sequence regardless if actual data is missing.

C. Stegofile is Injected at End of File

The method of hiding at the EOF, also known as the injection method, is a technique that directly appends the stegofile at the end of the cover file [12]. In this study, when a stegofile was injected at the EOF of the carrier FLV, the stegoFLV played the entire length normally and without distortions. The injection method was successful in hiding the stegofile into the carrier while retaining the audio and video quality of the carrier. However, this method can significantly increase the size of the stegoFLV, depending on the size of the stegofile hidden, which is the sum of the file sizes of the carrier and the stegofile. Virtually, unlimited amount of data can be hidden in an FLV by using the injection method, but hiding a

large file will make the stegoFLV suspicious because of its large file size. Moreover, the stegoFLV is prone to suspicion and attack because the stegofile is just dangling at the EOF. In some cases, parts of the stegofile, such as the file extension, file type, and even the actual contents, can be read when opened in a hex editor.

D. Stegofile is Hidden at the Metadata

When the stegofile was hidden at the metadata and corresponding changes were made in the metadata's body size and previous tag size, the stegoFLV played the entire length normally and without distortions. However, similar to the injection method, this method can also significantly increase the size of the carrier. Nevertheless, an interesting result was achieved when the entire metadata was replaced with the actual stegofile, and corresponding changes were made in its body size and previous tag size. The stegoFLV also played the entire length normally and without distortions. Although in this case, the file size of the stegoFLV is not the sum of the file sizes of the cover and the stegofile but is rather the sum of the file sizes of the cover and the stegofile minus the size of the metadata of the cover. Even though this method successfully hid the stegofile, it is not recommended because adding data or altering the metadata is prone to suspicion and attack because the metadata usually contains only useful information about the FLV itself, such as the duration of the video, framerate, video and audio data rates, and codec used, among others. If important metadata contents are modified, especially upon the encoding of the FLV, abnormalities in the playback may be encountered, including the seekbar being not in sync with the video itself [13], or seeking to a specific time of the playback is not possible. The typical size of a metadata tag is below 1,000 bytes, and thus, hiding a relatively large data in the metadata would be suspicious. Moreover, many third party applications can alter the metadata for encoding, possibly altering and even removing the hidden stegofile.

E. Stegofile is Hidden Before or Inside the Actual Data of a Video Tag

When the stegofile was hidden before or inside the actual data of a video tag, the stegoFLV played the entire length but with distortions on the timestamp of the modified video tag. These methods are not recommended because obvious distortions will be encountered, making it prone to suspicions and attacks.

F. Stegofile is Hidden at the End of a Video or Audio Tag

When the stegofile was hidden at the end of an audio tag, just after the actual data of the tag and before the corresponding previous tag size, and assuming that corresponding changes were made in the audio tag's body size and previous tag size, the stegoFLV played the entire length but with distortions in the timestamp of the modified audio tag. For example, the stegofile was appended at the 4,049 ms timestamp of an audio tag with a size of 523 bytes. During playback of the stegoFLV, a split-second pause in the audio was encountered somewhere in the 4-second mark. Thus, this method is not recommended because the stegoFLV will encounter distortions and will be prone to suspicion and attack. On the other hand, when the stegofile was hidden at the end of a video tag, just after the actual data of the tag and before its corresponding previous tag size, and assuming that the corresponding changes were made in the video tag's body size and previous tag size, the stegoFLV played the entire length normally and without noticeable distortions. For example, the stegofile was appended at the 4,037 ms timestamp of a video tag with a size of 725 bytes. During playback of the stegoFLV, no distortions were encountered throughout the entire video, even at the 4-second mark.

Using this method, the stegofile is hidden at a video tag inside the body of the carrier, which is less suspicious and more difficult to locate for an attacker. Even when a considerably large chunk of data (75,720 bytes) was hidden, the quality of the stegoFLV during playback did not encounter any noticeable distortions based on the auditory–visual perception tests. However, this method produces the similar file size issue as the injection method. In addition, if an attacker maps the sizes of the video tags of the stegoFLV, a large spike will be noticed, especially if the size of the stegofile is considerably larger than the sizes of the video tags of the carrier. For example, the stegoFLV described above and the carrier were analyzed using the automated FLV analysis software and the graphs of their video tags evolution were generated, as shown in Figs. 5 and 6, respectively. A large spike in the graph of the stegoFLV is noticeable when compared with that of the carrier.

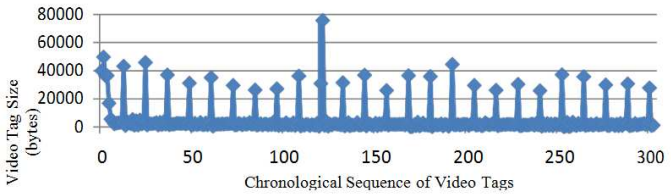

Figure 5. Graph of the video tags evolution of the stegoFLV.

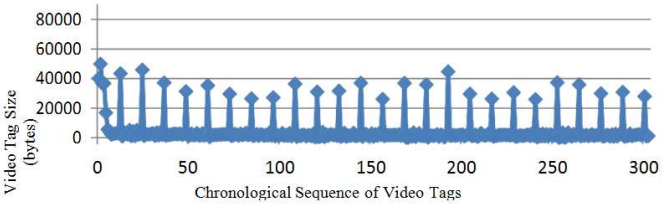

Figure 6. Graph of the video tags evolution of the carrier.

G. Stegofile is Distributed Among All of the Video Tags

When the stegofile was divided into smaller pieces and then hidden among all of the video tags of the carrier, just after the actual data of each video tag and before its corresponding previous tag size, and assuming that the corresponding changes were made in each tag's body size and previous tag size, the stegoFLV played the entire length normally and without any noticeable distortions. For example, using the automated FLV analysis software, the carrier was found to have a total of 303 video tags. Thus, the stegofile was divided into 303 parts and hidden among all of the video tags. This method aims to make

the stegoFLV even less suspicious compared with the methods described above by dividing the stegofile into smaller pieces and hiding them at different portions of the body of the carrier. Imagine dividing a paragraph composed of 100 words and writing each word on different but specific pages of a book. Even if a reader notices a suspicious word on a page, the entire secret remains secure because the other words that would complete the hidden paragraph are located at the other pages. Moreover, even if an attacker graphs the video tags evolution of the stegoFLV, the graphs would not be suspicious because the graph of the stegoFLV replicates the shape of the graph of the carrier (which is the usual pattern of video tags evolution of different canonical videos analyzed in a separate experiment not discussed in this study). The only difference that can be seen is that the sizes of all video tags shifted upwards pertaining to the sizes of the added pieces of the stegofile. Fig. 7 shows a superimposed graph of the video tags evolutions of the stegoFLV and the carrier (only a small portion of the complete graph is shown for zoom).

A disadvantage of this method is that when the histogram of the video tags evolutions of the stegoFLV is generated, a shift in the Gaussian-like curve to the right will be noticed because of the sizes of the added pieces of the stegofile on each video tag (especially when the stegofile is large). For example, a rar/zip file, which contained 26 files with a total size of 2,128,405 bytes, was hidden inside an FLV, which had a file size of 2,523,885 bytes and contained 268 video tags. Fig. 8 shows the histograms of the video tags of the cover and the stegoFLV.

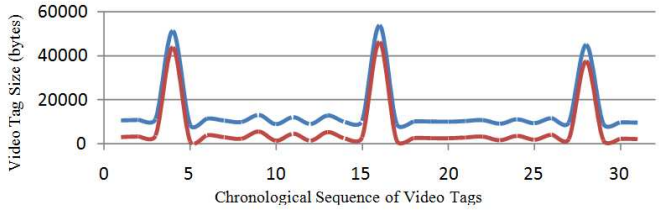

Figure 7. Superimposed graphs of the video tags evolutions of the stegoFLV (top) and carrier (bottom).

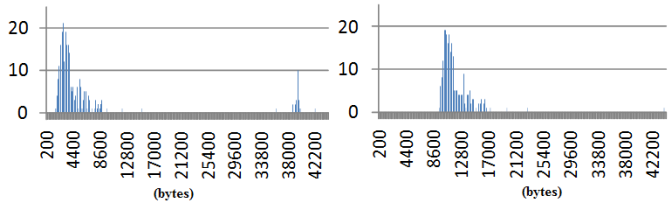

Figure 8. Histograms of the video tags evolutions of the cover (left) and stegoFLV (right).

To further test the video quality of the stegoFLVs, the ROI_RGB_averaging software was used. A carrier, the stegoFLV with distortions, and the stegoFLV with no noticeable distortions were analyzed, as shown in Fig. 9. The ROI_RGB_averaging software calculates the RGB histogram of a small box positioned inside the video frames at a specific part of the videos. As can be seen in Figure 9, (a) and (c) are almost similar, while (b) has a noisier graph, which corresponds to the blotting of the shirt of the walking boy. Thus, it can be concluded that the steganographic methods that do not generate noticeable distortions in the stegoFLV will not be suspicious when the RGB averages are mapped.

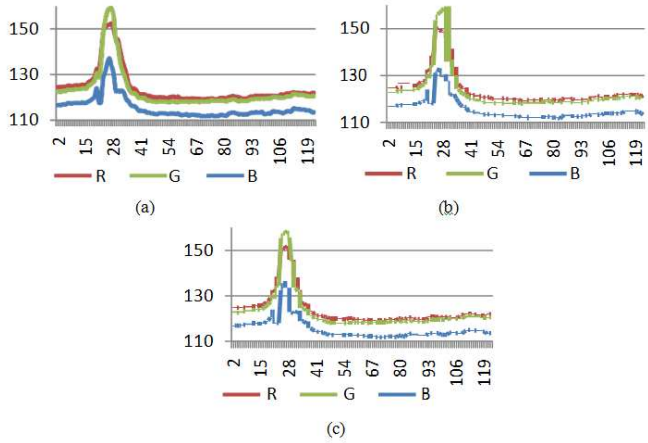

Figure 9. RGB averages of the (a) carrier, (b) stegoFLV with distortions, and (c) stegoFLV without distortions.

H. Tags are Removed to Compensate for the Size of the Added Stegofile

To compress the carrier and compensate for the size of the stegofile to be hidden, video tags would have to be deleted. Deleting audio tags to compress the carrier is not recommended because they are very small in size, usually below 600 bytes, and noticeable split-second distortions in the audio will be generated. As discussed above, removing the video tags would cause distortions on the video quality at the timestamps of the removed tags. Thus, the use of FLVs that are monochromatic, or with minimal movements, or even with still images that act as a video is recommended to successfully compress the carrier without generating noticeable distortions in the video quality. The distortions in such videos will not be noticeable compared with the videos with moving objects, such as the video of the walking boy, and with noisy backgrounds possibly because a current frame in an FLV is dependent on the previous frames. To verify the statements above, a video of a gray background with a total size of 1,689,693 bytes was compressed. First, the video was analyzed using the automated FLV analysis software and the graph of its video tags evolution was generated, as shown in Fig. 10.

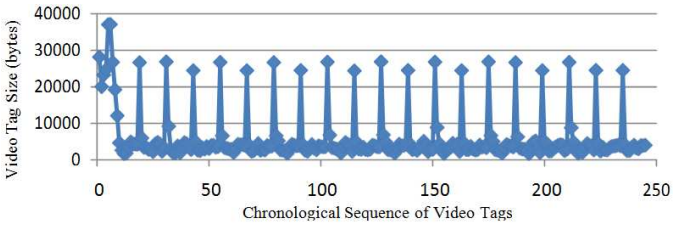

Figure 10. Graph of the video tags evolution of the original gray video.

To show a noticeable compression, the spikes on the graph or the video tags with sizes greater than 10,000 bytes (excluding the first video tag) were removed. The modified

FLV, now with a total size of 1,001,676 bytes only, was compressed in size by almost 40% compared with the original FLV. During playback, the degradation in quality or the distortions of the modified cover compared with the original carrier were not noticeable at all. Thus, an effective compression can be applied on FLVs that are monochromatic, with minimal movements, or with still images that act as a video. Compression can even be applied, for example, on an FLV with a combination of monochromatic, minimal movements, still images, moving subjects, or noisy sceneries, as long as the removal of the tags, or the compression, is applied at specific timestamps where the distortions will not be noticeable (i.e., the characteristics of the FLVs where effective compression can be applied as demonstrated above).

I. Embed/Extract Stegofile in/from an FLV Carrier

The methods above were applied by hiding a stegofile inside an FLV carrier. The stegoFLV was sent and received through electronic mail, file-sharing websites, infrared, and Bluetooth, and the recipient was able to extract the stegofile without loss. The extracted data was verified whether it was modified through the process it underwent by comparing its file hash with that of the stegofile embedded [14]. A file hash is used for verifying that a file has not been tampered with when downloaded or extracted. The file hashes of the embedded and extracted stegofiles matched.

V. Conclusions

The current study successfully demonstrated a solution to the current issues of security for digital records through the application of steganography. The proposed effective steganographic methods are as follows: (1) Injection at the EOF; (2) Embedding at a video tag inside the FLV; (3) Embedding at the metadata; and (4) Distributing the stegofile among all of the video tags. Each method has its corresponding strengths and weaknesses, and the stegoFLVs created using the proposed techniques were analyzed qualitatively by using auditory–visual perception tests and quantitatively by using video tags evolution graphs and histograms and RGB averaging analysis.

The methods of hiding the stegofile at the end of a video tag and distributing the stegofile among all of the video tags of a carrier are recommended as the most effective steganographic methods for hiding information inside an FLV carrier because of the following reasons: (1) No noticeable distortions can be observed in the video and audio quality of the stegoFLV, making it look like a typical FLV out in the open; (2) The stegofile is hidden inside the FLV and not at the EOF or header, making it less prone to suspicion or attack; (3) The stegofile may not be altered even if the stegoFLV is converted to a different file format and when its header or metadata is changed or removed; (4) The ways to check for suspicions (which are the video tags evolutions and histograms) should be accompanied by software, such as the automated FLV analysis software, which is difficult to do for a typical third party or observer who gets hold of the stegoFLV.

If a user wants to address the size of the added stegofile, he/she can use and compress videos that are monochromatic, with minimal movements, or with still images that act as a video. In such cases, even if the user deletes some of the video tags across the FLV, no noticeable distortions will be observed in the modified carrier. Moreover, the size of the stegoFLV with the hidden stegofile can even be smaller than that of the carrier. The suite of tools, including the automated FLV analysis and the video steganography software, was essential in performing the experiments above and in embedding and extracting any type and any number of data within an FLV carrier.

Acknowledgment

This study was supported by the Philippine Government's Engineering Research and Development for Technology (ERDT).

References

[1] L. McGill. (2005, May 5). *Steganography: the right way.* [online]. Available: http://www.sans.org/reading_room/whitepapers/stenganography/steganography_1584. [Accessed: June 2011].

[2] K. Brown. (2006, Jan). *Security briefs: encrypting without secrets.* Available: http://msdn.microsoft.com/en-us/magazine/cc163676.aspx, [Accessed: June 2011].

[3] DotNetZip Library. Available: http://dotnetzip.codeplex.com/.

[4] B. Dunbar. (2003, Oct). *A detailed look at steganographic techniques and their use in an open-systems environment.* Available: http://www.sans.org/reading_room/whitepapers/covert/detailed-steganographic-techniques-open-systems-environment_677, [Accessed: June 2011].

[5] P. Hayati, V. Potdar, and E. Chang. *A survey of steganographic and steganalytic tools for the digital forensic investigator,* Available: http://www.pedramhayati.com/images/docs/survey_of_steganography_and_steganalytic_tools.pdf, [Accessed: June 2011].

[6] D. Salomon, *Data privacy and security*, Springer-Verlag New York, Inc., 2003.

[7] A. Mozo, M. Obien, C. Rigor, D. Rayel, K. Chua, G. Tangonan, "Video steganography using Flash Live Video (FLV)," in *Instrumentation and Measurement Technology Conference*, Singapore, 2009.

[8] E. Villegas, private communication, 2011.

[9] S. Adee, "Spy vs. Spy," IEEE Spectrum: The magazine of technology insiders, August 2008.

[10] HHD Software Free Hex Editor Neo, Available: http://www.hhdsoftware.com/free-hex-editor.

[11] Flash Video (FLV), Available: http://osflash.org/flv.

[12] Zone-H.Org, *Steganography FAQ,* Available: http://infosecwriters.com/text_resources/pdf/Steganography_AMangarae.pdf, [Accessed: June 2011].

[13] A. Basu, *FLV metadata*, Available: http://www.easyflv.com/kb/flv-metadata.php, [Accessed: June 2011].

[14] HashTab. Available: http://implbits.com/HashTab.aspx.