International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT217352

Data Hiding and Retrieval Method Using LSB Substitution Algorithm

Asoke Nath*[1], Soham Mondal[1], Raj Deb[1], Akash Das[1]
[1]Department of Computer Science, St.Xavier's College, Kolkata, West Bengal, India

Article Info
Volume 7, Issue 3
Page Number: 267-279

Publication Issue :
May-June-2021

Article History
Accepted : 15May2021
Published : 22May2021

ABSTRACT

Steganography is the art of hiding the fact that communication is taking place, by hiding information in other information. Many different carrier file formats can be used, but digital images are the most popular because of their frequency on the Internet. For hiding secret information in images, there exists a large variety of steganography techniques some are more complex than others and all of them have respective strong and weak points. Different applications have different requirements of the steganography technique used. For example, some applications may require absolute invisibility of the secret information, while others require a larger secret message to be hidden. This project hides the message within the image. This project uses LSB Substitution method to hide the secret message inside the image. Sender select the cover image with the secret message file (which can be of any format) and hide it into the image, it help to generate the secure stego image. The stego image is sent to the destination with the help of private or public communication network. On the other side the receiver downloads the stego image and using the software retrieve the secret text hidden in the stego image.

Keywords: Steganography, Digital images.

I. INTRODUCTION

The word Steganography is a Greek word means "covered, or hidden writing".

Steganography is a special method of writing hidden messages in such a way that no-part from the sender and the receiver can even realize that there is a hidden message. Today, the term Steganography includes the embedding of digital information within computer files. For example, the sender may embed a big text file in an image in such a way that there should not be any significant change in the image. We can embed even some voice or image or text in any host file which may be an image file or may be another sound file etc.

The reality is that secret communication is used for a variety of reasons and by a variety of people. For example the business people when sending some important information from one place to another place can hide the actual information in some cover file which may be a simple file. In defense also the Steganography has a very important role, protecting company trade secrets. Governments hide information from other Governments, and technophiles amuse themselves by sending secret messages to each other just for fun. The only tie that

binds all these people is a desire to hide something from someone else.

Steganography is the art of hiding the fact that communication is taking place, by hiding information in other information. Many different carrier file formats can be used, but digital images are the most popular because of their frequency on the Internet. For hiding secret information in images, there exists a large variety of Steganographic techniques, some are more complex than the others and all of them have respective strong and weak points. Different applicants have different requirements and hence different Steganography techniques are used. For example, some applications may require absolute invisibility of the secret information, while others require a larger secret message to be hidden.

In the present work the authors have introduced a new method for hiding any encrypted secret message inside a cover file. The cover file is chosen to be an image file about 10 times larger than the payload file. For encrypting secret message the authors have introduced a unique byte modification algorithm. The technique used here is little bit modified as according to the traditional method of modifying the LSBs of every pixels. Instead of changing the LSBs of the cover file at a stretch the authors proposed to change the LSB's of the Red, Green and Blue channel of every pixel. By doing so the advantage is that maximum amount of data can be hidden within given cover image.

In the present method the authors applied the algorithm on different standard payload files such as image file, audio file, video file, Microsoft word file, Excel file, Power point file, .exe file and in every case the result found was satisfactory. The authors found that to embed any payload file we need to select a cover image file whose size is much more than the payload file and the size depends on the image file types. For example if .jpg or .png files are chosen as the cover file then the payload file of maximum size can be about 90% of the size of the cover file. But if .bmp file is chosen as the cover image file then the payload file size should be less than 10% of the size of the cover image file. This is because .jpg or .png files are compressed initially so the actual file size is much larger than it's compressed version and thus it can embed such huge amount of data but the problem that has been noticed by the authors is that the stego file obtained using .jpg file is about 10 times larger in size as compared to its compressed version. This problem has been overcome by using .bmp files which is devoid of any kind of compression.

In order to keep the process simple as well as intangible for the intruders to obtain the actual hidden message a keyless method is used to encrypt the data. The position of the message bits have been shifted by certain value making the task difficult for the intruder to obtain the actual values. Moreover there is no possibility for the intruder to find out the starting pixel and the ending pixel which has been modified to embed the secret message.

II. HISTORY OF STEGANOGRAPHY

The earliest recordings of Steganography were by the Greek historian Herodotus in his chronicles known as "Histories" and date back to around 440 BC. Herodotus recorded two stories of Steganographic techniques during this time in Greece. The first stated that King Darius of Susa shaved the head of one of his prisoners and wrote a secret message on his scalp. When the prisoner's hair grew back, he was sent to the Kings son in law Aristogoras in Miletus undetected. The second story also came from Herodotus, which claims that a soldier named Demeratus needed to send a message to Sparta that Xerxes intended to invade Greece. Back then, the writing medium was text written on wax-covered tablets. Demeratus removed the wax from the tablet, wrote the secret message on the underlying wood, recovered the tablet with wax to make it appear as a

blank tablet and finally sent the document without being detected.

Romans used invisible inks, which were based on natural substances such as fruit juices and milk. This was accomplished by heating the hidden text, thus revealing its contents. Invisible inks have become much more advanced and are still in limited use today.

During the 15th and 16th centuries, many writers including Johannes Trithemius(author of Steganographia) and Gaspari Schotti (author or Steganographica) wrote on Steganographic techniques such as coding techniques for text, invisible inks, and incorporating hidden messages in music.

Between 1883 and 1907, further development can be attributed to the publications of Auguste Kerckhoff (author of Cryptographic Militaire) and Charles Briquet (author of Les Filigranes). These books were mostly about Cryptography, but both can be attributed to the foundation of some Steganographic systems and more significantly to watermarking techniques.

During the times of WWI and WWII, significant advances in Steganography took place. Concepts such as null ciphers (taking the 3rd letter from each word in a harmless message to create a hidden message, etc), image substitution and microdot (taking data such as pictures and reducing it to the size of a large period on a piece of paper) were introduced and embraced as great Steganographic techniques.

In the digital world of today, namely 1992 to present, Steganography is being used all over the world on computer systems. Many tools and technologies have been created that take advantage of old Steganographic techniques such as null ciphers, coding in images, audio, video and microdot. With the research this topic is now getting we will see a lot of great applications for Steganography in the near future.

III. STEGANOGRAPHY METHODS

There are a large number of Steganographic methods that most of us are familiar with (especially if you watch a lot of spy movies!), ranging from invisible ink and microdots to secreting a hidden message in the second letter of each word of a large body of text and spread spectrum radio communication. With computers and networks, there are many other ways of hiding information, such as:

1. Covert channels (e.g., Loki and some distributed denial-of-service tools use the Internet Control Message Protocol, or ICMP, as the communications channel between the "bad guy" and a compromised system)
2. Hidden text within Web pages
3. Hiding files in "plain sight" (e.g., what better place to "hide" a file than with an important sounding name in the *c:|winnt|system32* directory?)
4. Null ciphers (e.g., using the first letter of each word to form a hidden message in an otherwise innocuous text)

Steganography today, however, is significantly more sophisticated than the examples above suggest, allowing a user to hide large amounts of information within image and audio files. These forms of Steganography often are used in conjunction with cryptography so that the information is doubly protected; first it is encrypted and then hidden so that an adversary has to first find the information (an often difficult task in and of itself) and *then* decrypt it.

There are a number of uses for Steganography besides the mere novelty. One of the most widely used applications is for so-called *digital watermarking*. A watermark, historically, is the replication of an image, logo, or text on paper stock so that the source of the document can be at least partially authenticated. A digital watermark can accomplish the same function; a graphic artist, for example,

might post sample images on her Web site complete with an embedded signature so that she can later prove her ownership in case others attempt to portray her work as their own.

Steganography can also be used to allow communication within an underground community. There are several reports, for example, of persecuted religious minorities using Steganography to embed messages for the group within images that are posted to known Web sites.

The following formula provides a very generic description of the pieces of the steganographic process:

cover_medium + hidden_data = stego_medium

In this context, the *cover_medium* is the file in which we will hide the *hidden_data*, which may also be encrypted. The resultant file is the *stego_medium* (which will, of course. be the same type of file as the cover_medium). The cover_medium (and, thus, the stego_medium) are typically image or audio files. In this article, I will focus on image files and will, therefore, refer to the *cover_image* and *stego_image*.

Before discussing how information is hidden in an image file, it is worth a fast review of how images are stored in the first place. An image file is merely a binary file containing a binary representation of the color or light intensity of each picture element (pixel) comprising the image.

Images typically use either 8-bit or 24-bit color. When using 8-bit color, there is a definition of up to 256 colors forming a palette for this image, each color denoted by an 8-bit value. A 24-bit color scheme, as the term suggests, uses 24 bits per pixel and provides a much better set of colors. In this case, each pixel is represented by three bytes, each byte representing the intensity of the three primary colors red, green, and blue (RGB), respectively. The Hypertext Markup Language (HTML) format for indicating colors in a Web page often uses a 24-bit format employing six hexadecimal digits, each pair representing the amount of red, blue, and green, respectively. The color orange, for example, would be displayed with red set to 100% (decimal 255, hex FF), green set to 50% (decimal 127, hex 7F), and no blue (0), so we would use "#FF7F00" in the HTML code.

The size of an image file, then, is directly related to the number of pixels and the granularity of the color definition. A typical 640x480 pix image using a palette of 256 colors would require a file about 307 KB in size (640 · 480 bytes), whereas a 1024x768 pix high-resolution 24-bit color image would result in a 2.36 MB file (1024 · 768 · 3 bytes).

To avoid sending files of this enormous size, a number of compression schemes have been developed over time, notably Bitmap (BMP), Graphic Interchange Format (GIF), and Joint Photographic Experts Group (JPEG) file types. Not all are equally suited to steganography, however.

GIF and 8-bit BMP files employ what is known as *lossless* compression, a scheme that allows the software to exactly reconstruct the original image. JPEG, on the other hand, uses *lossy*compression, which means that the expanded image is very nearly the same as the original but not an exact duplicate. While both methods allow computers to save storage space, lossless compression is much better suited to applications where the integrity of the original information must be maintained, such as steganography. While JPEG can be used for stego applications, it is more common to embed data in GIF or BMP files.

The simplest approach to hiding data within an image file is called *least significant bit (LSB) insertion*. In this method, we can take the binary representation of the hidden_data and overwrite the LSB of each byte within the cover_image. If we are using 24-bit color, the amount of change will be minimal and indiscernible to the human eye. As an example, suppose that we have three adjacent pixels (nine bytes) with the following RGB encoding:

10010101 00001101 11001001
10010110 00001111 11001010
10011111 00010000 11001011

Now suppose we want to "hide" the following 9 bits of data (the hidden data is usually compressed prior to being hidden): 101101101. If we overlay these 9 bits over the LSB of the 9 bytes above, we get the following (where bits in **bold** have been changed):

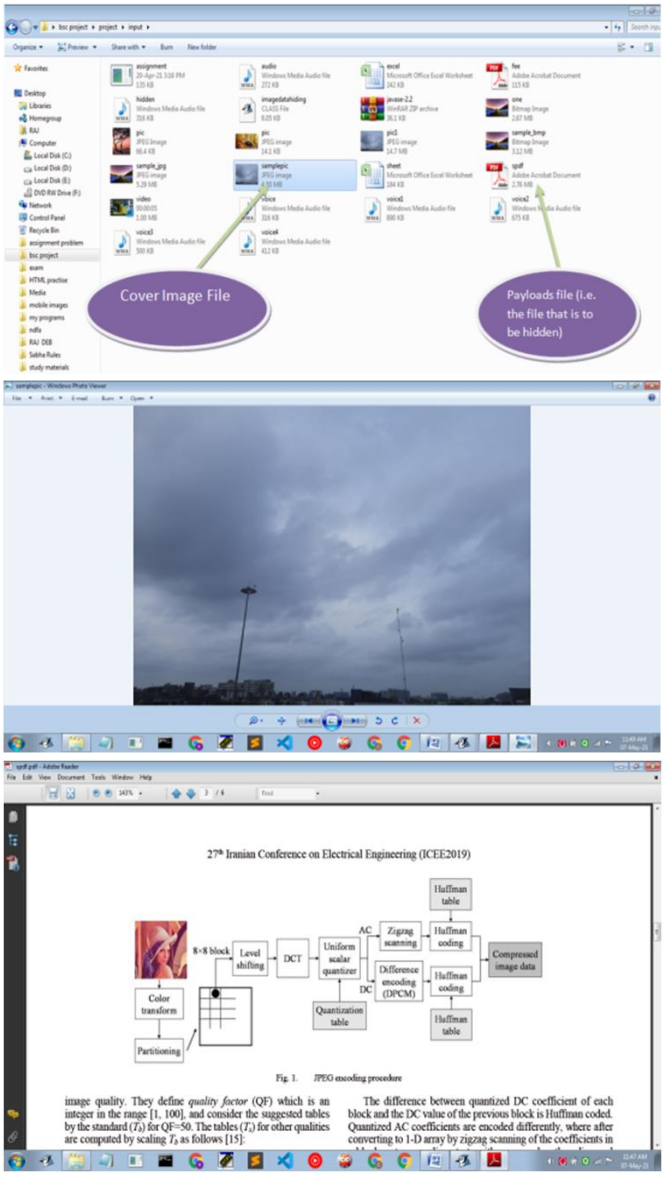
10010101 00001100 11001001
10010111 00001110 11001011
10011111 00010000 11001011

Note that we have successfully hidden 9 bits but at a cost of only changing 4, or roughly 50%, of the LSBs.

This description is meant only as a high-level overview. Similar methods can be applied to 8-bit color but the changes, as the reader might imagine, are more dramatic. Gray-scale images, too, are very useful for steganographic purposes. One potential problem with any of these methods is that they can be found by an adversary who is looking. In addition, there are other methods besides LSB insertion with which to insert hidden information.

Without going into any detail, it is worth mentioning *steganalysis*, the art of detecting and breaking steganography. One form of this analysis is to examine the color palette of a graphical image. In most images, there will be a unique binary encoding of each individual color. If the image contains hidden data, however, many colors in the palette will have duplicate binary encodings since, for all practical purposes, we can't count the LSB. If the analysis of the color palette of a given file yields many duplicates, we might safely conclude that the file has hidden information.

But what files would you analyze? Suppose I decide to post a hidden message by hiding it in an image file that I post at an auction site on the Internet. The item I am auctioning is real so a lot of people may access the site and download the file; only a few people know that the image has special information that only they can read. And we haven't even discussed hidden data inside audio files! Indeed, the quantity of potential cover files makes steganalysis a Herculean task.



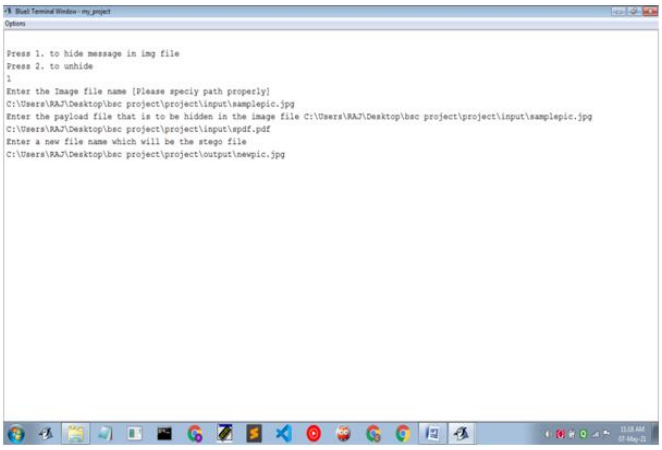
In the above figures samplepic.jpg is chosen to be the cover image file and spdf.pdf is chosen to be the payload file (i.e. the secret file).

The algorithm takes input a cover image file and a payload file and after processing it returns the stego file which is similar to the cover image file while viewing with bare eyes but in reality few bits of pixels have been modified. Now we are going to explain the process step by step.

1. Input the files:

In this step the user has to input the name of the cover image file and the payload file. The user has to give the path and file names properly.

Internally a byte array stores all the bytes of the inputted payload file.



2. Modifying the bytes of the payload file

The raw data bytes that are stored in the byte array is encrypted by just modifying the data bytes as follows. All the byte values which are in even indices of the array is added with 12 and all the byte values which are in odd indices of the array is subtracted by 12.

By doing such modification we ensure the security of the hidden file. Since if we assume that the intruder has successfully revealed the payload file from the stego file then he will not get the original hidden file but a modified one which will be actually useless to him.

3. Creating File overheads

File overheads include the size of the payload file and the extension of the file. We can obtain file length using a java function and the file extension is obtained from the variable which stores the name of the payload file that was inputted by the user in step 1.

Along with the message bits we need to also store the file overheads information which is actually required while extracting the payload file from the stego file.

Since this is crucial information so instead of storing the data as it is we do little bit of modification as follows:

We exchange the first 4 characters with the last 4 characters and subtract 12 from the ASCII values of this 8 characters. Then from position m-1 to m+1 we add 14 to the ASCII values of the these 3 characters where m=(total no. of characters-1)/2.

For example:

Suppose the user inputs a payload file sample.pdf **(100KB).**

File overhead="**102400E.pdf**"

Here 100*1024=102400B is the file size and .pdf is the file extension and 'E' is the separator.

Modification takes place as follows:

File Overheads	Description
10 2400E.pdf	This is the original one
%$&(00E"dXZ	First 4 characters is subtracted by 12. Similarly last 4 characters are subtracted by 12.
"ZXd"00E(&$%	First 4 characters are exchanged with the last 4 characters.
ZXd">>S(&$%	From position m-1 to m+1 all the characters are added with 14. Here m is the (total no. of characters-1)/2.

Modified File overheads="SZXd">>S(&$%H".

S and H are added to the beginning and end of the string to indicate the start and end of the actual file overhead part.

This information is then converted into bytes and stored in a byte array.

4. LSB Substitution Technique

In order to understand how to modify the pixels of the cover image we first understand a pixel.

A digital image is stored as a 2D array of pixels and a pixel is the smallest element of digital image.

The 2D array of pixels can be viewed as follows:

P11	P12	…….	P1n
P21	P22	…….	P2n
.			
.	.		
.	.		
.	.		
.			
.			
.	.		
.	.		
.	.		
.			
Pm1	Pm2	……	Pmn

Fig.2 (m X n)

Here m=height of the cover file and n=width of the cover file

P_{ij}= Pixel value (4 bytes) at i^{th} row and j^{th} column

Each pixel contains the values of alpha, red, green, blue values and the value of each color lies between 0 to 255 which consumes 8 bits (2^8).

The ARGB values are stored in 4 bytes of memory in the same order (right to left) with blue value at 0-7 bits,

Green value at 8-15 bits, Red value at 16-23 bits and, alpha at 24-31 bits.

We can retrieve the RGB values from a pixel using the shift operators .To do so –

- Right, shift to the beginning position of each color i.e. 24 for alpha 16 for red, etc.
- The shift right operation may impact the values of other channels, to avoid this, you need to perform bitwise and operation with 0Xff. This masks the variable leaving the last 8 bits and ignoring all the rest of the bits.

```
p = img.getRGB(x, y);
//Getting the A R G B values from the pixel value
a = (p>>24)&0xff;
r = (p>>16)&0xff;
g = (p>>8)&0xff;
b = p&0xff;
```

Using the below example it is clear that from a pixel we can obtain four 8 bit values corresponding to its respective positions in the pixel.

Random Pixel P_{ij}	**(P_{ij}>>24) & 0XFF (alpha)**	**(P_{ij}>>16 & 0XFF (red)**	**(P_{ij}>>8) & 0XFF (green)**	**P_{ij} & 0XFF (blue)**
11111101	00000000	00000000	00000000	11111101
11101010	00000000	00000000	11111101	11101010
10011001	00000000	11111101	11101010	10011001
11001101	11111101	11101010	10011001	11001101
	&	&	&	&
	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000
	11111111	11111111	11111111	11111111
	=	=	=	=
	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000
	11111101	11101010	10011001	11001101

Now the algorithm has all the components required for embedding the secret payload file in the cover image file. It has the file overheads byte array obtained in step 3, it has the raw data bytes of the payload file obtained in step 2 and finally, it has the R,G,B and A values corresponding to every pixels of the cover image.

We replace the LSB's of R,G and B values of some of the pixels of the cover image with the message bits until all the elements of the byte array which stored

the data bytes is utilized and using this modified R, G and B values we create a new image which is the stego image. The stego image will be almost similar to the cover image since only one bit of R, G and B values is modified which is impossible to be distinguished with bare eyes. We do not modify the Alpha values.

As discussed earlier in fig 2, corresponding to every coordinate(i,j) we get a pixel value. The authors have decided to start replacing the pixel values from a position i=0, j=312. This position is arbitrarily chosen. And the file overheads are also embedded at position i=10. While the rest of the bits are modified with the payload file bits.

The following figure will help us to understand which pixels are actually modified to embed the payload file along with its overheads.

$P_{0,0}$	$P_{0,1}$	…….	$P_{0,312}$	……	$P_{0,n}$
$P_{1,0}$	$P_{1,1}$	…….	……	……	$P_{1,n}$
$P_{2,0}$	$P_{2,1}$	…….	……	……	$P_{2,n}$
. .	. .	. .	. .	. .	. .
$P_{10,0}$	$P_{10,1}$	…….	……	……	$P_{10,n}$
. .	. .	. .	. .	. .	. .
$P_{m,0}$	$P_{m,1}$	…….	……..	…….	$P_{m,n}$

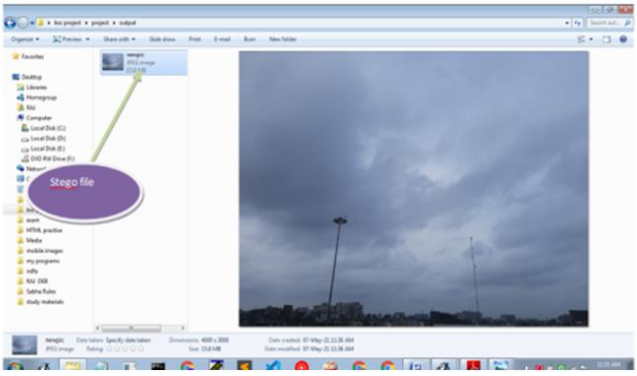
From the above figure the shaded colored portion represents that the pixels at these positions can be modified with the payload file bits. And the blue colored portion represents that the pixels at these positions is to be modified for the file overhead bits. Whereas the unshaded portion is left as it is.

5. Creating Stegonographed Image

In step 4 we have obtained R, G, B and A values corresponding to every pixel and also modified the LSBs of those values with the payload file bits and it's overhead. In this step we are going to combine the R, G, B and A values in order to form the modified pixels. Then this pixels are combined together and written in the hard disk with a new file name as inputted by the user in step 1.This is the stego image.

```
finalpixel=(a<<24);
finalpixel=finalpixel | (r<<16);
finalpixel=finalpixel | (g << 8);
finalpixel=finalpixel | b;
```

This is the process how we can create a new pixel with the given R, G, B and A values.



The above figure shows that upon successful running of the algorithm a stego file which is similar to the cover file is created at the location specified by the user in step 1.

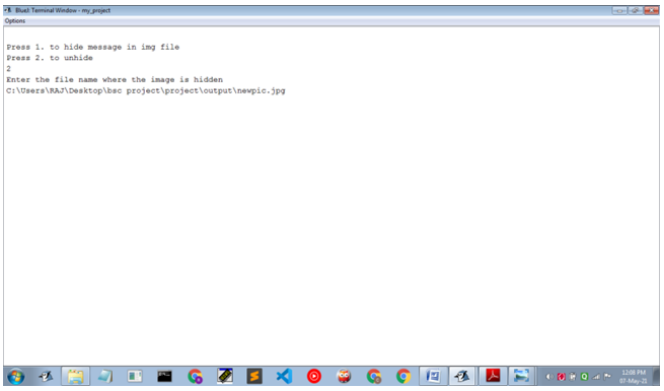
6. Reveal the hidden file

The receiver on receiving the stego file at his end through the public communication medium will now feed in the algorithm with name of the stego image by mention it's path clearly and the algorithm will extract the hidden file from the stego image and save the file in the same location as the stego file. If there is no hidden message file present in the image

received then the program will display the appropriate message.

a) Input from user

The user will input the name of the stego file along with it's path.



b) Extracting the file overheads of the hidden file

In step 3 of the previous algorithm we have discussed the process to create the file overheads of the payload file and in the step 4 we have also discussed that in position i=0 of the 2D array of pixels we have modified the LSBs of these file overhead bits. Now, in this step we are going to extract the LSB's of these pixels and covert every 8 bits into a character and store it in a string variable.

So following the example taken in step 3 of the previous algorithm we will get the file parameters (or file overhead) by just reversing the process as follows:-

File overhead="SZXd">>S(&$%H".

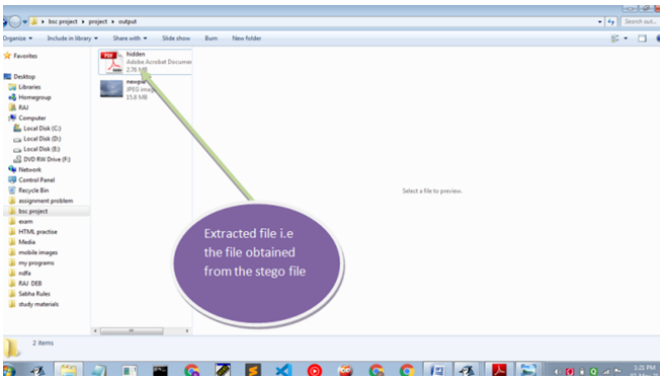
File overhead="ZXd">>S(&$%". Removing the last two variables as it was used just to mark the beginning and ending of the file overheads. If the variable S and H is missing then we will display the message "There is no hidden file" and the program terminates.

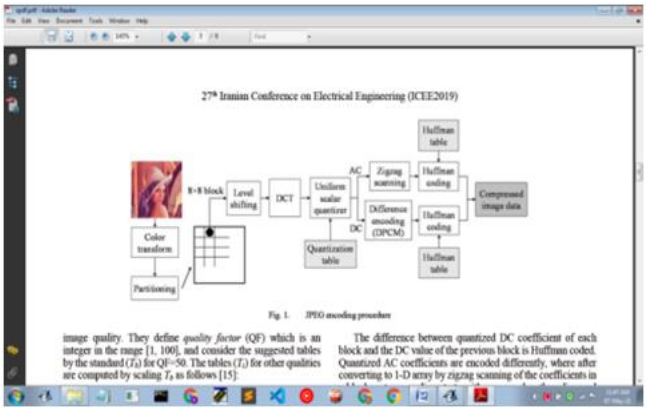
File Overheads	Description
ZXd">>S(&$%	This is the one obtained from the stego image.
"ZXd"00E(&$%	From position m-1 to m+1 all the characters are subtracted with 14. Here m is the (total no. of characters-1)/2.
%$&(00E"dXZ	First 4 characters are exchanged with the last 4 characters.
102400E.pdf	First 4 characters is added with 12. Similarly last 4 characters are added with 12.

Here we got the original file overheads back. Now with this value we can extract the data bytes of the payload file and save the file in the user inputted location.

c) Extracting the payload file bits

We have seen in Fig 3 that shows which pixels are actually modified to store the message file bits. We get the LSBs of only those pixels and we store every 8 bits in a byte array. As a result we get a byte array of size same as that is obtained in the file overheads in step 2 and every element of this byte array is actually the message bytes. These message bytes are then saved into the location as inputted by the user in step1.





From the above figures it is clear that the payload file spdf.pdf and the extracted file hidden.pdf is same.

IV. RESULTS

We applied our present method on different cover files and secret message files and the results are given below

• Case-1: Cover File type=.jpg secret message file =.jpg

Fig_1:Cover file name Fig_2:Secret message Fig_3: Embedded Cover file

(Secret message encrypted before embedding)

• Case-2: Cover File type=.BMP secret message file =.doc

Fig_4: Cover File Fig_5: Secret message Fig_6: Embedded Cover file

(Secret message encrypted before embedding)

• Case-3: Cover File type=.BMP secret message file =.jpg

Fig_7: Cover file Fig_8: Secret message Fig_9: Embedded cover file

(The secret message was encrypted before embedded)

• Case-4: Cover File type=.jpeg secret message file =.pdf

Fig_10: Cover File Fig_11: Secret message Fig_12:Embedded Cover File

(The encrypted secret message file is embedded)

V. CONCLUTION

In the present work we try to embed some secret message inside cover image file in encrypted form so that no one will be able to extract actual secret message. The program is developed in JAVA. We embed the data in the LSBs of the R, G and B component of every pixel of the cover file. The bytes of the secret message file are encrypted before embedding into the cover image file so that the task becomes difficult for the intruder to decrypt the actual hidden message. Moreover we left first few pixels unchanged and position of actual modification of the pixel starts from a fixed pixel value which is kept secret, the advantage of this is that the even if the intruder gets a partial portion of the file, the file will get corrupted. Moreover file extension and file size are most important factors which is also encrypted in a different way and this Information is stored in the cover image file in few specific pixels which only the algorithm knows, thus it is almost undetectable by the intruder.

VI. HARDWARE AND SOFTWARE REQUIREMENTS

No as such specific Hardware add-ons are required to compile the following program other then the default System Requirements to execute JAVA executable files. This was build using JAVA and with the support of Java Directory Kit (JDK) version 12.0.1 and is supportable to Java Runtime Environment (JRE) version 12.0.1.

VII. ACKNOLEDGEMENT

On the very outset, We would like to express our sincere gratitude to our project advisor Dr. Ashoke Nath, Associate Professor, St. Xavier's College, Kolkata, for his continuous support in our project. His guidance supporting our ideas with patience, motivation, enthusiasm and immense knowledge, helped us all the time through our work. We could not have imagined having a better advisor and mentor for our project. We came to know about so many new things from him, and are really grateful to him for that.

We also take this opportunity to thank all the professors of our department and to all our friends who have shared their valuable inputs with us and helped us in making this project a success.

Last but not the least; we are indebted to our parents who have always supported us morally and economically.

VIII. REFERENCES

[1]. Data Hiding and Retrieval, A. Nath, S. Das, A. Chakrabarti, Proceedings of IEEE. International conference on Computer Intelligence and Computer Network held at Bhopal from 26-28 Nov, 2010.

[2]. Advanced Steganographic approach hiding encrypted secret message in LSB,LSB+1, LSB+2and LSB+3 bits in non standard cover files, Joyshree Nath, Sankar Das, Shalabh Agarwal and Asoke Nath , International Journal of Computer Applications (0975-8887) Vol 14-No7, Feb 2011.

[3]. Advanced Steganography Algorithm using Encrypted secret message, Joyshree Nath and Asoke Nath, International Journal of Advanced Computer Science and Application (IJACSA) Vol-2 No.3 March (2011) (Accepted for publication).

[4]. Symmetric key cryptography using random key generator, A. Nath, S. Ghosh, M.A. Mallik, Proceedings of International conference on SAM-2010 held at Las Vegas(USA) 12-15 July, 2010, Vol-2, P-239-244.

[5]. Steganography in Digital Media Principles, Algorithms, and Applications- Jessica Fridrich, Cambridge University Press, 2010.

[6]. Jpeg20000 Standard for Image Compression Concepts algorithms and VLSI Architectures by Tinku Acharya and Ping-Sing Tsai, Wiley Interscience.

[7]. Steganography and Seganalysis by Moerland, T, Leiden Institute of Advanced Computing Science.

[8]. An Overview of Image Steganography by T.Morkel, J.H.P. Eloff and M.S.Oliver.

[9]. Websites (Tutorialspoint, Geeksforgeeks, Wikipedia), and other online resources.

IX. AUTHOR PROFILE

Dr. Asoke Nath is working as Associate Professor in the Department of Computer Science, St. Xavier's College (Autonomous), Kolkata. He is engaged in research work in the field of Cryptography and Network Security, Steganography, Green Computing, Big data analytics, Li-Fi Technology, Mathematical modelling of Social Area Networks, MOOCs etc. He has published 253 research articles in different Journals and conference proceedings.

Mr. Soham Mondal is a student of St. Xavier's College, currently pursuing B.Sc. in Computer Science. His interests lie in the field of Cyber Security, Data Science, AI and real world project implementation of these fields.

Mr. Raj Deb is a student of St. Xavier's College, currently pursuing B.Sc. in Computer Science. His interests lie in the field of Cyber Security, Data Science, Algorithms, Coding, Cryptography, AI and real world project implementation of these fields.

Mr. Akash Das is a student of Comp[uter Science in St. Xavier's College. His interests lie in the field of Cryptanalysis, Data Models, DBMS and real world project implementation of these fields.

Cite this article as :

Asoke Nath, Soham Mondal, Raj Deb, Akash Das, "Data Hiding and Retrieval Method Using LSB Substitution Algorithm", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 7 Issue 3, pp. 267-279, May-June 2021. Available at doi : https://doi.org/10.32628/CSEIT217352
Journal URL : https://ijsrcseit.com/CSEIT217352