Информационные процессы, Том 7, № 1, 2007, стр. 33–44.

Application of Cryptography and Randomized Spatial Domain Steganography for Information Hiding in Medical Images

Srihari Sridharan

B.Tech Information Technology,
BSA Crescent Engineering College (Affiliated to Anna University),
Vandalur, Chennai – 600 048, India
srhariin03@yahoo.com

Received January 13, 2006

Abstract—Steganography is the art and science of writing hidden messages in such a way that no one apart from the intended recipient knows of the existence of the message; this is in contrast to cryptography, where the existence of the message itself is not disguised, but the content is obscured. Medical information about the patient is very sensitive and security solutions are needed to make the information secure during transmission and distribution. This paper proposes an image steganography as a means to hide this secret information inside the image. The proposed method ensures that there is minimal change in the medical image since any degradation will reduce the clinical value of the image. The proposed method encodes the data into the image in a randomized manner. This paper proposes a framework for data embedding using hashing, encryption and randomization, where in other hashing, encryption and randomization techniques can be plugged into the model as per security needs.

1. INTRODUCTION

The medical images are playing an important role in making optimal diagnosis. Clinicians analyze the medical images and add annotations and opinions. The existing e-diagnosis system is discussed in [XQJ2003], the paper also proposes an e-diagnosis scheme. In order to obtain a second opinion from a physician in a distant area, he should have the privilege to access the system by Internet for the patient images and reports. It may be impractical to let many outside physicians have their accounts to access the system. This increases the load on the system and would hurt patient information security.

In the proposed system, the clinician analyses the image, records the annotations and diagnosis comments, embeds the details into the image, and then transmits the image through the Internet to another clinician. Image and data are integrated into a single unit. The data that is to be embedded into the image is first compressed using a compression algorithm. It is then encrypted and embedded into the image using the proposed techniques. The encryption procedure and the data embedding procedure use pseudorandom generator and hashing algorithm which can be replaced by standard or custom implementations. This paper provides a framework for data encryption and embedding using hashing and randomization. The paper also proposes a protocol format for embedding files of any format into the image. The data embedding procedure works on the basis of the protocol format.

2. OVERALL FUNCTIONALITY

The proposed system functionality is illustrated in Fig. 1. Physician A views the image from the image repository and diagnoses the image. He records his diagnosis comments and annotations. This information can be stored in a file or set of files. The file(s) is (are) first compressed and then encrypted. Then the encrypted files can be embedded inside the image using the data embedding procedure. The proposed protocol format is used for data embedding inside the image. The output of the data embedding process is the stego image which is transmitted over the internet to other physicians. This image is given as input to the data extraction procedure and the data is extracted from the image. Then the data is given to the decryption algorithm and the compressed file is obtained, which is passed on to the decompression algorithm and the output is the file(s) containing the diagnosis comments entered by Physician A. The system uses symmetric keys, i.e. the keys used for encryption / decryption and embedding / extracting are same. One advantage of embedding multiple files is that, the keys used for encryption can be hidden as a file along with other files.

The sender (Physician A) compresses and encrypts the secret information to be transmitted using secret key K1 (crypto key) and number of key encryption cycles N1. The encrypted data is embedded inside the cover image using secret key K2 (stego key) and number of key encryption cycles N2. The stego image is transmitted through the communication network and at the receiver end the data is extracted using the secret key K2 and the number of key encryption cycles N2. Then the secret key K1 and the number of key encryption cycles N1 is used to decrypt the cipher text. The original information is obtained after decompression. The proposed algorithms for cryptography, steganography and the protocol format are explained in detail in section 3.

3. PROPOSED ALGORITHMS FOR CRYPTOGRAPHY AND STEGANOGRAPHY

3.1. Cryptographic Algorithm

The inputs for the encryption procedure are plain text, crypto key and the number of encryption cycles for the crypto key encryption. The output is the cipher text. The inputs for the decryption procedure are cipher text, crypto key and the number of encryption cycles for the crypto key encryption. The output is the plain text.

3.1.1. Encryption Procedure

The plain text ‘P’ is converted into the cipher text ‘C’ using the key ‘K’ and the number of encryption cycles for the crypto key encryption ‘N’.

The steps involved in the encryption procedure are as follows:

1. The key ‘K’ is converted into a 128 bit offset value using MD5 hashing algorithm. This 128 bit value is used as seed to initialize a pseudorandom generator and the key is encrypted by performing a XOR operation between every byte in the key and a randomly selected value between 0 and 255. The entire key is encrypted in a similar manner.

2. The encryption of the key takes place for ‘N’ cycles specified by the user. The encrypted key generated in each cycle is converted into a 128 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

3. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 128 bit offset. The block size for file encryption is equal to key size i.e. 128 bits.

4. The plain text is converted into cipher text of same size in a two stage process. The plain text is first converted into intermediate cipher text and then the intermediate cipher text is converted into cipher text.

5. In the first stage there are various methods for generating the intermediate cipher text out of which one method is selected at random. The 128 bit offset is used for:

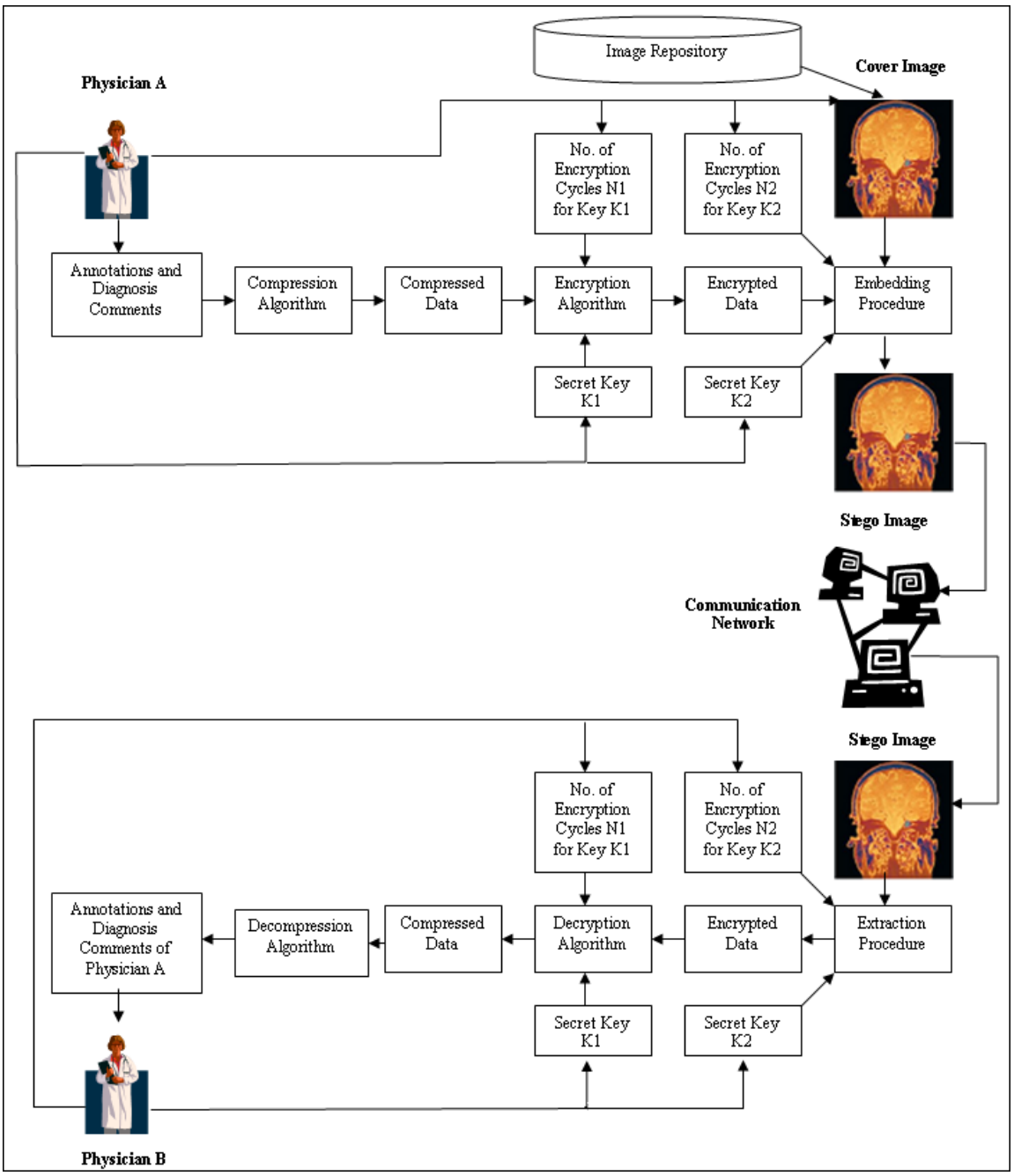


Fig. 1. Proposed system workflow

(i) Initializing the pseudorandom generator before one of the method is selected at random.
(ii) Initializing the pseudorandom generator for the conversion of intermediate cipher text to cipher text.
(iii) Converting the plain text to intermediate cipher text by performing XOR operation.
(iv) Initializing the pseudorandom generator for any shuffling operation to be performed after splitting the key during intermediate cipher text generation process.

6. The proposed methods of intermediate cipher text generation are as follows. The 128 bit offset is divided into 16 bytes (16 parts). The 128 bit offset is XOR-ed with data in various ways to generate the intermediate cipher text. They are as follows:

(i) Each byte is XOR-ed with a byte of plain text to get a 128 bit result. The 128 bit offset is cyclically right shifted by 8 bits and again XOR-ed with the 128 bit result obtained from the previous operation. The same operation is repeated for 16 iterations.

(ii) Each byte is XOR-ed with a byte of plain text to get a 128 bit result. The 128 bit offset is right shifted (non cyclic) by 8 bits and again XOR-ed with the 128 bit result obtained from the previous operation. The same operation is repeated for 16 iterations.

(iii) Each byte is XOR-ed with a byte of plain text to get a 128 bit result. The 128 bit offset is cyclically left shifted by 8 bits and again XOR-ed with the 128 bit result obtained from the previous operation. The same operation is repeated for 16 iterations.

(iv) Each byte is XOR-ed with a byte of plain text to get a 128 bit result. The 128 bit offset is left shifted (non cyclic) by 8 bits and again XOR-ed with the 128 bit result obtained from the previous operation. The same operation is repeated for 16 iterations.

(v) Shuffle 16 parts of the offset and XOR with the plain text to get a 128 bit result. Shuffle 16 parts of the offset each and every time by reinitializing the pseudorandom generator with the shuffled offset used in the previous iteration. The same operation is repeated for 16 iterations.

(vi) Shuffle 16 parts of the offset and XOR with the plain text to get a 128 bit result. Shuffle 16 parts of the offset each and every time (without reinitializing the pseudorandom generator). The same operation is repeated for 16 iterations.

(vii) Shuffle 16 parts of the offset and XOR with the plain text to get a 128 bit result. The 128 bit offset is cyclically right shifted by 8 bits and again XOR-ed with the 128 bit result obtained from the previous operation. The same operation is repeated for 16 iterations.

(viii) Shuffle 16 parts of the offset and XOR with the plain text to get a 128 bit result. The 128 bit offset is right shifted (non cyclic) by 8 bits and again XOR-ed with the 128 bit result obtained from the previous operation. The same operation is repeated for 16 iterations.

(ix) Shuffle 16 parts of the offset and XOR with the plain text to get a 128 bit result. The 128 bit offset is cyclically left shifted by 8 bits and again XOR-ed with the 128 bit result obtained from the previous operation. The same operation is repeated for 16 iterations.

(x) Shuffle 16 parts of the offset and XOR with the plain text to get a 128 bit result. The 128 bit offset is left shifted (non cyclic) by 8 bits and again XOR-ed with the 128 bit result obtained from the previous operation. The same operation is repeated for 16 iterations.

(xi) Each byte of plain text is XOR-ed with randomly selected value between 0 and 255 to get a 128 bit result. The 128 bit offset is cyclically right shifted by 8 bits and used to reinitialize the pseudorandom generator for next iteration. The same operation is repeated for 16 iterations.

(xii) Each byte of plain text is XOR-ed with randomly selected value between 0 and 255 to get a 128 bit result. The 128 bit offset is right shifted (non cyclic) by 8 bits and used to reinitialize the pseudorandom generator for next iteration. The same operation is repeated for 16 iterations.

(xiii) Each byte of plain text is XOR-ed with randomly selected value between 0 and 255 to get a 128 bit result. The 128 bit offset is cyclically left shifted by 8 bits and used to reinitialize the pseudorandom generator for next iteration. The same operation is repeated for 16 iterations.

(xiv) Each byte of plain text is XOR-ed with randomly selected value between 0 and 255 to get a 128 bit result. The 128 bit offset is left shifted (non cyclic) by 8 bits and used to reinitialize the pseudorandom generator for next iteration. The same operation is repeated for 16 iterations.

(xv) Each byte of plain text is XOR-ed with randomly selected value between 0 and 255 to get a 128 bit result. Each byte of the 128 bit result is XOR-ed with randomly selected value between 0 and 255 (without reinitializing the pseudorandom generator). The same operation is repeated for 16 iterations.

There are many other operations similar to those mentioned above to generate the intermediate cipher text. For example, more combinations can be formed by repeating some of the above steps

by reinitializing the random number generator between every cycle of XOR operation. Based on the security requirements other ways can be added to increase the security, as the increase in number of ways used to generate the intermediate cipher text increases the complexity involved in breaking the cipher increases.

7. Each byte in the intermediate cipher text is XOR-ed with a randomly selected number between 0 and 255. This byte is converted into ASCII value. In this way 128 bit cipher text for the given 128 bit plain text is obtained.

8. This block of cipher text is written into a file and the next block of plain text is obtained for encryption. The encrypted key is again encrypted and the 128 bit offset value is again computed. This is used to initialize the pseudorandom generator and used as the key for next block of plain text.

Thus the entire file is encrypted. If the file size is not an exact multiple of the block size then the last block is less than 128 bits in size. The required number of bits in the key is used for the last block. The key length decides the block size for encryption and also the number of iterations in the XOR-ing process during the intermediate cipher text generation process.

3.1.2. Decryption Procedure

The cipher text 'C' is converted into the plain text 'P' using the key 'K' and the number of encryption cycles for the crypto key encryption 'N'.

The steps involved in the decryption procedure are as follows:

1. The key 'K' is converted into a 128 bit offset value using MD5 hashing algorithm. This 128 bit value is used as seed to initialize a pseudorandom generator and the key is encrypted by performing a XOR operation between every byte in the key and a randomly selected value between 0 and 255. The entire key is encrypted in a similar manner.

2. The encryption of the key takes place for 'N' cycles specified by the user. The encrypted key generated in each cycle is converted into a 128 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

3. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 128 bit offset. The pseudorandom generator generates the same sequence of random numbers when it is initialized with the same seed value. The block size for file decryption is same as the key size i.e. 128 bits.

4. The cipher text is converted into plain text in a two step process. The first step involves the conversion of the cipher text to intermediate cipher text and then the intermediate cipher text is converted into plain text.

5. The first block of cipher text is read from the file. The 128 bit offset obtained from the encrypted crypto key is used to initialize the pseudorandom generator. Each byte in the cipher text is XOR-ed with a randomly selected number between 0 and 255. This gives the intermediate cipher text.

6. The next step is the conversion of intermediate cipher text to plain text. The conversion requires the computation of all the 128 bit offsets used for 16 iterations in advance, because the XOR operation between the intermediate cipher text and the 128 bit offset must performed in the reverse order. Consider the following equation, A XOR B gives C and C XOR D gives E, i.e.,

$$\boxed{\text{C = A XOR B, E = C XOR D}}$$

To get back A from E the following operations must be performed, E XOR D gives C and C XOR B gives A, i.e.

$$\boxed{\text{C = E XOR D, A = C XOR B}}$$

Thus when the same seed value is used for initializing the pseudorandom generator it generates the same sequence of random numbers used during the encryption process. The same method of intermediate cipher text generation process gets selected and the corresponding reverse operation is performed to get the plain text from the intermediate cipher text. In this way the 128 bit plain text block is obtained from the 128 bit cipher text block.

7. Then this plain text block is written into a file and the next bock of cipher text is obtained for decryption. The encrypted key used for previous block is again encrypted and the 128 bit offset value is again computed. This is used for decrypting the next block of cipher text. As during encryption if the last block size is less than 128 bits then required number of bits in the key are used. Thus the entire file is decrypted.

In the proposed algorithms MD5 hashing algorithm is used for hashing, this can be replaced by other algorithms like SHA-1, SHA-256, etc can be used. As the key size varies the number or XOR iterations in the intermediate cipher text computation and the block size of the data varies.

3.2. Steganographic Algorithm

The data embedding and extraction method describes how multiple files of any format are hidden inside an image. The inputs for embedding data are cover image, the files that are to be embedded inside the cover image, stego key and the number of encryption cycles for stego key encryption. The output is stego image. The inputs for the data extraction procedure are stego image, stego key and the number of encryption cycles for stego key encryption. The output is set of files present inside the stego image.

The fields present in the protocol format used for data embedding and extraction are explained below:

The **PAH** – *Password Authentication Header* is the SHA-1 hash of the password that is used for data embedding. The password entered by the receiver is hashed and compared with the PAH, only when both are same the receiver can extract the files from the image. The PAH is 160 bits long.

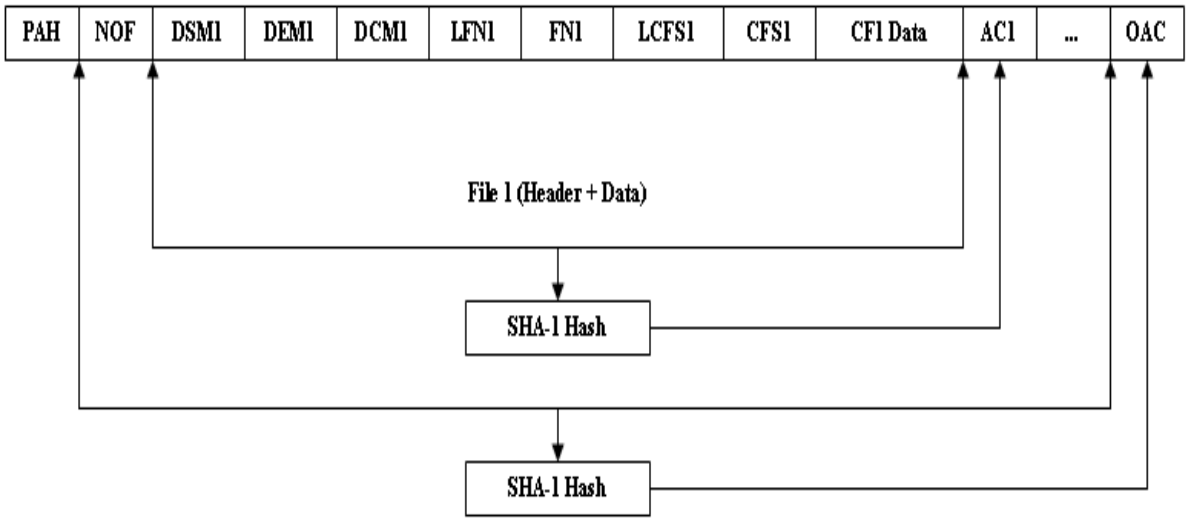
The **NOF** – *Number of Files* indicates the number of files stored inside the image. Maximum of 255 files is supported at present.

After embedding these details each and every file's data is embedded along with particular header information. The header contains information pertaining to data storage mode, data encryption mode, data compression mode, file size, file name and format information.

The **DSM n** – *Data Storage Mode of* n^{th} *file* indicates the method of data embedding in the LSB, whether it is 1 LSB, 2 LSBs or either of the 2 LSBs at random. A pixel is selected at random and the pixel value can be used in case of 8 bit grayscale bitmap and one among the RGB components present in the pixel can be used at random in case of a 24 bit bitmap. At present 4 modes are proposed. These are 1 LSB for grayscale image and 1 LSB or 2 LSBs or either of 2 LSBs for 24 bit bitmaps. The number of LSBs used for data storage is proportional to the image distortion. The DSM for a particular file is selected at random. This field is 8 bits long.

The **DEM n** – *Data Encryption Mode of* n^{th} *file* indicates the encryption algorithm used for file encryption. The proposed encryption can be used along with provision for other cryptographic algorithms. The DEM for a particular can be standardized or be selected at random from a set of algorithms. This field is 8 bits long.

The **DCM n** – *Data Compression Mode of* n^{th} *file* indicates the compression technique used for file compression. The file is compressed before encryption. Compression aids encryption by reducing the redundancy of the plaintext. If an encryption algorithm is good, it will produce output which is statistically indistinguishable from random numbers and no compression algorithm will successfully



Field Name	Description	Length (in Bits)
PAH	Password Authentication Header. (SHA-1 Hash of the password)	160
NOF	Number of files stored inside the image. (Maximum 255 Files)	8
DSM n	Data Storage Mode for n^{th} file – Indicates how the data is stored in the LSB of the pixel component.	8
DEM n	Data Encryption Mode for n^{th} file – Indicates the cryptographic algorithm.	8
DCM n	Data Compression Mode for n^{th} file – Indicates the compression algorithm.	8
LFN n	Length of file name (with extension) of n^{th} file.	8
FN n	File name (with extension) of n^{th} file.	Length * 8
LCFS n	Length of file size of n^{th} compressed file.	8
CFS n	File size of n^{th} compressed file. (File size represented in Hexadecimal. 4 bits per hexadecimal digit.)	Length * 4
CF n Data	Compressed file data of n^{th} file.	File Length * 8
AC n	Authentication Checksum for n^{th} file. (SHA-1 Hash computed over n^{th} file header and data)	160
OAC	Overall Authentication Checksum. (SHA-1 Hash computed over n^{th} file header and data)	160

Fig. 2. Protocol Format for Data Embedding and Extraction

compress random numbers. On the other hand, if a compression algorithm succeeds in finding a pattern to compress out of an encryption's output, there is a flaw in the encryption algorithm.

The **LFN n** – *Length of File Name of* n^{th} *File* indicates the number of characters present in the file name including the extension. Suppose the file name is 'abc.txt' then LFN is 7. This field is 8 bits long. If the file name with extension is greater than 255 characters the rightmost 255 characters are taken since the extension needs to be preserved.

The **FN n** – *File name of* n^{th} *File* indicates the filename along with extension. Files of various formats have different headers. To reduce the header processing overhead the file extension information is stored along with the filename, since the operating system uses the extension to decide the application used to open the file.

The **LCFS n** – *Length of Compressed File Size of* n^{th} *File* indicates the number of digits present in the hexadecimal representation of the file size. This field is 8 bits long.

The **CFS n** – *File Size of* n^{th} *Compressed File* indicates the file size represented in hexadecimal. If the file size is 1024 B then the hexadecimal representation is 400, the LCFS will be 3. Since each

hexadecimal digit needs 4 bits, 1 byte can hold 2 digits. In case of odd number of digits a zero is appended in the front, for e.g. 400 gets converted as 0400 and it is stored in 2 bytes.

The **CF n Data** – *Data in* n^{th} *Compressed File* represents the compressed file data.

The **AC n** – *Authentication Checksum for file n* is the SHA-1 hash computed over *DSM, DEM, DCM, LFN, FN, LCFS, CFS and CF Data* of a file. This ensures the data integrity of the file and its header. This field is 160 bits in length.

The **OAC** – *Overall Authentication Checksum* is the checksum computed over all the fields except the *PAH – Password Authentication Header*. This ensures overall data integrity.

The *PAH, NOF, OAC* and the *DSM* fields are embedded using 2 LSBs. Other fields are embedded based on the *DSM* of a particular file.

(a) Data embedding procedure

The cover image can be gray scale bitmap (8 bit or 24 bit) or 24-bit true color bitmap image. The stego image obtained is same as the cover image. The set of files 'F' are embedded inside cover image 'C' using the key 'K' and the number of encryption cycles for the stego key encryption 'N' there by producing the stego image 'S' as output.

The steps involved in the data embedding procedure are as follows:

1. The key 'K' is converted into a 128 bit offset value using MD5 hashing algorithm. This 128 bit value is used as seed to initialize a pseudorandom generator and the key is encrypted by performing a XOR operation between every byte in the key and a randomly selected value between 0 and 255. The entire key is encrypted in a similar manner.

2. The encryption of the key takes place for 'N' cycles specified by the user. The encrypted key generated in each cycle is converted into a 128 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

3. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 128 bit offset and the pseudorandom generator is initialized. The PAH, NOF are embedded first and the pixel locations where these information is stored is maintained in a set 's' this is done to avoid reusing the used pixel and color component pair.

4. The key is encrypted and the pseudorandom generator is reinitialized,

(i) Before embedding the PAH and NOF fields.
(ii) Before embedding the DSM, DEM, DCM, LFN, FN, LCFS and CFS fields.
(iii) Before embedding every block of CF data and AC.
(iv) Before embedding the OAH.

5. The block size is 128 bits. The first block of data to be embedded is taken and embedded inside the cover image as follows:

(i) Every block of data is embedded byte by byte.
(ii) The data is stored by using the least significant bit (LSB) or by using the two LSBs of the color component or by using either of the two LSBs at random. In case of 8 bit gray scale bitmaps the data is encoded in the LSB of the 8 bit pixel data. This is indicated by 'Mode' in the protocol format and it varies for every file that is embedded, for a particular file the mode is selected at random.
(iii) A byte of data is split into 8 parts of 1 bit each or 4 parts of two bits each based on the number of LSB used. Each part is stored in the LSB of the color component.
(iv) A pixel is selected at random by randomly selecting a row 'r' and column 'c'. In case of 24 bit images the color components present in the pixel in terms of red, green and blue are obtained. In case of 8 bit gray scale the 8 pixel data is obtained.

(v) In case of 24 bit images, a color component is selected at random and the data is embedded in the LSB. The color value is computed from the new color component values and restored. In case of 8 bit gray scale images the data is embedded in the LSB and the new color value is restored.

(vi) When a row, column and color component are selected at random the algorithm must ensure that the selected values are new since reusing the already used row, column and color component damages the already stored data.

(vii) This is handled by maintaining a set 's' that contains the already selected values. When new values of row, column and color component are selected at random they are compared with the values in the set. If the values are new then the data is stored in the selected location, else the algorithm finds an unused position.

(vii) Thus the data block is embedded into the image based on the protocol format.

5. The next block of data to be embedded is obtained. The encrypted key used for the previous block is again encrypted and the 128 bit offset value is again computed. This is used to initialize the pseudorandom generator before embedding the next block of data. If the data is not exact multiple of 128 bits then the last block has less than 128 bits. Thus the entire data is split into blocks and embedded into the image. (Other image formats such as JPEG and GIF can be used as cover images but needs to be converted to bitmap before embedding data).

(b) Data extraction procedure

The set of files 'F' are extracted from the stego image 'S' using the key 'K' and the number of encryption cycles for the stego key encryption 'N'.

The steps involved in the data extraction procedure are as follows:

1. The key 'K' is converted into a 128 bit offset value using MD5 hashing algorithm. This 128 bit value is used as seed to initialize a pseudorandom generator and the key is encrypted by performing a XOR operation between every byte in the key and a randomly selected value between 0 and 255. The entire key is encrypted in a similar manner.

2. The encryption of the key takes place for 'N' cycles specified by the user. The encrypted key generated in each cycle is converted into a 128 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 128 bit offset. The pseudorandom number generator is reinitialized in the same way as during data embedding, there by producing the same sequence of random numbers.

3. The PAH is extracted from the image, the receiver password is hashed using SHA-1 and compared with the PAH. If both are same then the data extraction is performed. The NOF gives us the number of files to be extracted. The DSM decides how the data is extracted. The DEM decides the cryptographic algorithm for decryption and the DCM decides the compression algorithm to be used for decompression of the extracted file. As in the embedding process a set 's' is maintained which contains the location from which data was extracted.

4. The same row, column and the color component which were selected during the data embedding process will be selected since the same seed is given to the pseudorandom generator. The data is extracted byte by byte until one block of data is extracted. The compressed file is extracted from the image block by block. One block of data is extracted byte by byte.

5. The encrypted key used for the previous block is again encrypted and the 128 bit offset value is again computed. This is used to initialize the pseudorandom generator before extracting the next block of data.

6. The entire data is extracted in a similar manner. The files are saved separately and their integrity is checked by comparing the computed checksum and original checksum. The OAC is

used for overall data integrity check. Then the files are decrypted and decompressed based on their DEM and DCM.

Thus multiple files can be embedded into and extracted from the image. One more way of embedding is selecting a byte of pixel data from the file at random leaving the image file header and color palette information (if any) present in the file. The same protocol format can be used in that case. This protocol format can also be used for data embedding/extraction in other media like audio and video. It can be used with other techniques viz. frequency domain techniques.

Data embedding / extraction based on Region of Interest (ROI) - In a medical image, the data can be encoded in the region other than the ROI. This is done in order to avoid the critical region of the image getting affected by the data embedding process. The ROI can be user defined or edge detection techniques can be used to identify the non-critical region and data can be embedded in that region. The image can be divided into many regions and data can be embedded redundantly inside the image to withstand attacks.

4. TEST RESULTS

The test result is shown below. 16 KB of data was embedded inside an image with dimensions 230 X 230 (24 bit color bitmap). The figure also shows the histogram of the color channels with the mean, standard deviation and median of color components before and after embedding data in the image. There is very little difference between the mean and the standard deviation values of the original and the stego image, indicating uniform distribution of data.

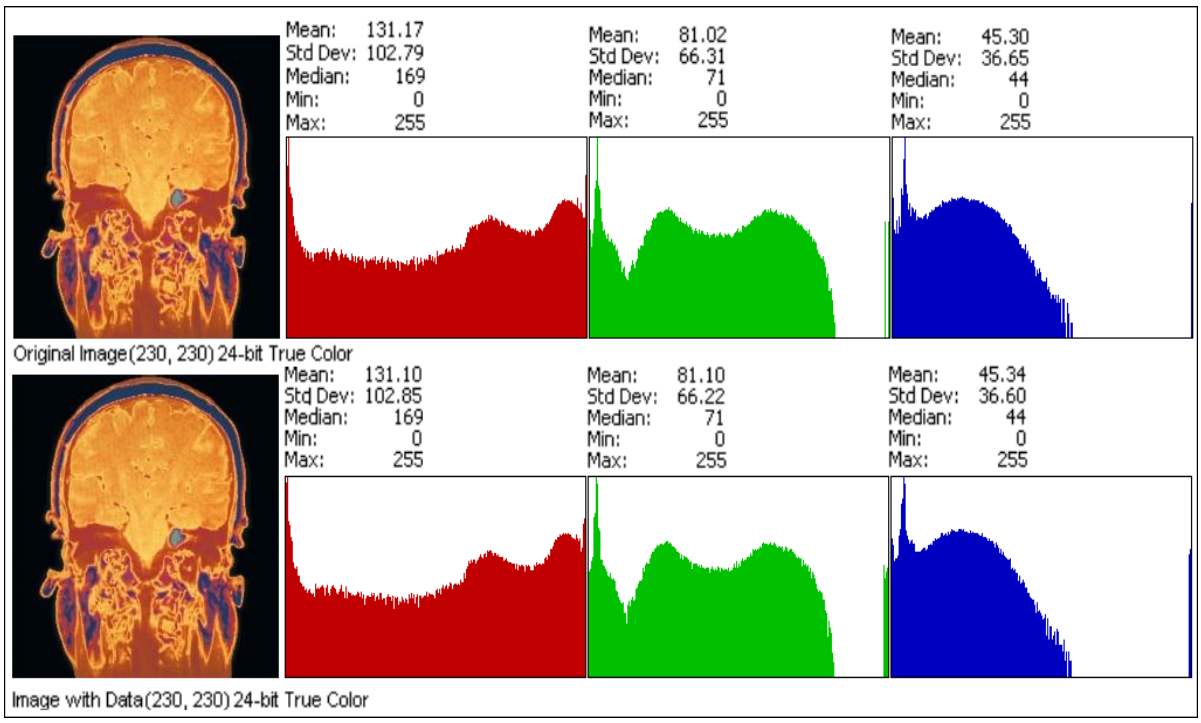


Fig. 3. Image Analysis

The Table 1 and graph in Fig. 4 below show the variation in Root Mean Square Error (RMSE) of the three color channels (RGB) between the original and the stego image with variation is amount of the embedded data, in 1 LSB and 2 LSB modes.

Dimension (230, 230) 24 Bit True Color Bitmap			Mode : 1 LSB Max Capacity : 19827 B			Mode : 2 LSB Max Capacity : 39655 B		
NOF	File Size (B)	Data + Header (B)	RMSE Red	RMSE Green	RMSE Blue	RMSE Red	RMSE Green	RMSE Blue
1	1000	1019	0.166	0.151	0.16	0.29	0.293	0.303
2	2000	2033	0.234	0.214	0.224	0.411	0.419	0.428
3	3000	3047	0.288	0.261	0.275	0.505	0.512	0.522
4	4000	4061	0.334	0.3	0.317	0.583	0.589	0.601
5	5000	5075	0.374	0.336	0.354	0.65	0.658	0.67
6	6000	6089	0.408	0.37	0.386	0.711	0.719	0.738
7	7000	7103	0.441	0.4	0.416	0.765	0.775	0.795
8	8000	8117	0.472	0.429	0.445	0.818	0.827	0.849
9	9000	9131	0.5	0.455	0.471	0.867	0.878	0.901
10	10000	10145	0.527	0.48	0.496	0.913	0.925	0.95
11	11000	11159	0.553	0.505	0.52	0.956	0.974	0.995
12	12000	12173	0.577	0.528	0.542	0.999	1.018	1.038
13	13000	13187	0.601	0.55	0.565	1.039	1.062	1.08
14	14000	14201	0.624	0.57	0.586	1.077	1.103	1.121
15	15000	15215	0.646	0.59	0.606	1.115	1.144	1.16
16	16000	16229	0.667	0.609	0.628	1.151	1.183	1.198
17	17000	17243	0.687	0.628	0.647	1.186	1.219	1.233
18	18000	18257	0.706	0.647	0.666	1.22	1.256	1.266

Table 1. File Size (in B) and Root Mean Square Error Values

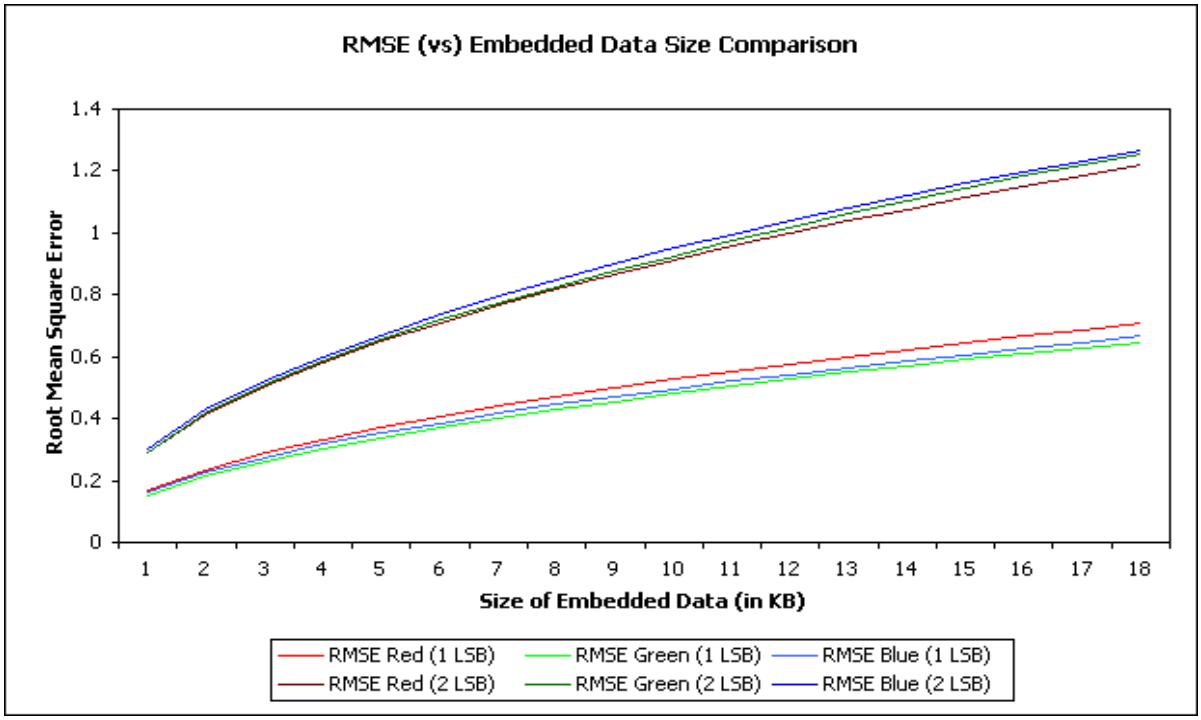


Fig. 4. Root Mean Square Error Comparison

5. CONCLUSION

The proposed method provides acceptable image quality with very little distortion in the image. It also provides provision for multiple files of any format to be embedded inside the image. The data is embedded in a randomized manner. The advantage of the proposed protocol format is that,

it can be used for data embedding in other media and also with other steganographic techniques. The proposed framework allows a plug and play feature for using any combination of cryptographic, hashing and random number generation algorithms.

6. ACKNOWLEDGEMENT

I would like to thank Prof. S. P. Reddy, former Head of the Department of Information Technology, BSA Crescent Engineering College, for his invaluable guidance, constant support and encouragement. I would like to thank my friends Venkatakrishnan R., Venkataraman R., Pratyusha R. and Karthikeyan R. for their useful discussions.

References

[GW92] Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Addison-Wesley. Reading, MA, (1992)

[XQJ2003] Xuanwen Luo, Qiang Cheng, Joseph Tan: A Lossless Data Embedding Scheme for Medical Images in Applications of e-Diagnosis.

[JJ98F] Johnson, N.F., Jajodia, S.: Exploring Steganography: Seeing the Unseen. IEEE Computer. February (1998) 26-34

[JJ98A] Johnson, N.F., Jajodia, S.: Steganalysis of Images Created Using Current Steganography Software. Proceedings of the Second Information Hiding Workshop held in Portland, Oregon, USA, April (1998)

[Sta99]Stallings, W.: Cryptography & Network Security: Principles and Practice, Prentice Hall, Upper Saddle River, New Jersey, (1999)

[Sch96] Schneier, B.: Applied Cryptography: Protocol, Algorithms, and Source Code in C, 2^{nd} Ed. New York, NY: John Wiley & Sons, (1996)

[EC2003] Cole, Eric: Hiding in Plain Sight: Steganography and the Art of Covert Communication, Wiley Publishing, Inc, (2003)