Информационные процессы, Том 7, № 1, 2007, стр. 13–32.

A Computationally Secure Randomized Method for Cryptography and Spatial-Domain Steganography

Srihari Sridharan

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

Received December 12, 2006

Abstract—Cryptography deals with the design of algorithms for encryption and decryption. In this paper a computationally secure cryptographic algorithm which uses 64-bit keys for encryption and decryption of data is proposed. Steganography is the art of passing information in a manner that the very existence of the message is unknown. The goal of steganography is to avoid drawing suspicion to the transmission of a hidden message. If suspicion is raised, then this goal is defeated. In this paper, the drawbacks in current spatial domain steganography software are listed and rectified. In this paper, a new data hiding method for storing secret data inside images and audio files in spatial domain is proposed. The proposed method stores data inside image and audio files using a pseudorandom generator, which makes detection of hidden information difficult, even if presence of hidden information is known. The proposed method doesn't require the aid of a cryptographic algorithm to make the data secure, although a cryptographic algorithm can be used for additional security. In the proposed method, the data to be embedded may be plain text or cipher text obtained by applying the plain text to a cryptographic algorithm. A password obtained from the user is used to initialize the pseudorandom generator. The location of the bytes where data is embedded is determined by the next value generated by the pseudorandom generator. The data is embedded in the least significant bit. The pseudorandom generator produces the same sequence of values when it is initialized with a particular seed. This fact is used for the decoding process, where the seed is generated from the password. A new procedure has been developed for data encoding without manipulating the cover media, based on its contents. In this method a log file is generated for the files that are to be secretly transmitted through the network. The log file is encrypted and sent along with the cover media to the other end. The cover media can be a file of any format.

1. INTRODUCTION

Cryptography is the branch of cryptology dealing with the design of algorithms for encryption and decryption, intended to ensure the secrecy and/or authenticity of messages. Steganography is the art of data hiding that conceals the existence of the secret messages in the media.

Cryptography deals with the design of algorithms for encryption and decryption. Cryptography provides a way to distribute files in secret code, or *cipher*, so intended recipients can only read them. An encryption scheme is **unconditionally secure** if the cipher text generated by the scheme does not contain enough information to determine uniquely the corresponding plain text, no matter how much cipher text is available. There is no encryption algorithm that is unconditionally secure. A cryptographic algorithm is **computationally secure** if the following two criteria are met.

(i) The cost of breaking the cipher exceeds the value of the encrypted information.
(ii) The time required to break the cipher exceeds the useful lifetime of the information.

Steganography encompasses methods of transmitting secret messages through innocuous cover carriers in such a manner that the very existence of the embedded messages is undetectable. Creative methods have been devised in the hiding process to reduce the visible detection of the embedded messages. An overview of current steganography software and methods applied to digital images is examined in [JJ98F].

Hiding information, where electronic media are used as such carriers, requires alterations of the media properties which may introduce some form of degradation. If applied to images that degradation, at times, may be visible to the human eye and point to signatures of the steganographic methods and tools used. These signatures may actually broadcast the existence of the embedded message, thus defeating the purpose of steganography, which is hiding the existence of a message. Images, audio and video files can be used as carriers.

2. TERMINOLOGY

Cryptography deals with encryption and decryption of data. A **Crypto Algorithm** is a procedure that takes the plain text data and transforms it into cipher text in a reversible way. **Plain text** refers to any message that is not encrypted. It is the input to an encryption function or the output of a decryption function. **Cipher text** refers to the data that has been encrypted. It is the output of an encryption function and the input to a decryption function. **Encryption** is the conversion of plain text or data into unintelligible form by means of a reversible translation, based on a translation table or algorithm. **Decryption** is the translation of encrypted text or data called cipher text into original text or data called plain text. **Password** or **Cryptokey** is a character string used to authenticate an identity. Knowledge of the password and its associated user ID is considered proof of authorization to use the capabilities associated with that user ID. **Pseudorandom Number Generator** is a function that deterministically produces a sequence of numbers that is apparently statistically random. A possible formula or process may be represented as: -

$$\boxed{\textit{Plain Text} + \textit{Cryptokey} = \textit{Cipher Text}}$$

Steganography literally means "covered writing"and is the art of hiding the very existence of a message. The possible **Cover Carriers** are innocent looking carriers (images, audio, video, text, or some other digitally representative code) which will hold the hidden information. A **Message** is the information hidden and may be plain text, cipher text, images, or anything that can be embedded into a bit stream. Together the cover carrier and the embedded message create a **Stego-Carrier**. Hiding information may require a **Stegokey** which is additional secret information, such as a password, required for embedding the information. A possible formula of the process may be represented as: -

$$\boxed{\textit{Cover-Medium} + \textit{Embedded Message} + \textit{Stegokey} = \textit{Stego-Medium}}$$

3. STEGANOGRAPHIC METHODS

The Internet is widely used as a channel for communication and information transfer, for both individual and mass communication. Images can be used as excellent carriers for hiding information and many such techniques have been discussed in [JJ98F] and a subset of steganography and digital watermarking tools available at present have been analyzed in [JJ98A]. There are two broad categories of these tools: Image Domain Tools and Transform Domain Tools.

Image Domain Tools use bit-wise methods that apply least significant bit (LSB) insertion and noise manipulation as the techniques for hiding data in images. There are some tools of this kind viz. StegoDos, S-Tools, Mandelsteg, EzStego, Hide and Seek, Hide4PGP, Jpeg-Jsteg, White Noise

Storm, and Steganos. The image formats typically used in such steganography methods are lossless and the data can be directly manipulated and recovered.

The Transform Domain Tools include those that involve manipulation of algorithms and image transforms such as discrete cosine transformation (DCT) and wavelet transformation. These tools hide the data in more significant areas of the image and may manipulate image properties such as luminance. The tools that belong to this group include PictureMarc, JK-PGS, SysCop, and SureSign [JJ98A]. These techniques are more robust than bit-wise techniques; however a tradeoff exists between the robustness obtained and the amount of information added to the image [JJ98F]. Many of these transform domain methods may survive conversion between lossless and lossy formats, as these methods are independent to image format.

The techniques like patchwork, pattern block encoding, spread spectrum methods [JJ98A] and masking [JJ98F] – which add redundancy to the hidden information - share characteristics of both image and transform domain tools. These techniques may help protect the information stored in the image when some image processing such as cropping and rotating is performed on the image. The patchwork approach uses a pseudo-random technique to select multiple areas (or patches) of an image for marking. Each patch may contain the watermark, so if one is destroyed or cropped, the others may survive. Masks may fall under the image domain as being an added component or image object.

4. DRAWBACKS IN THE CURRENTLY USED SOFTWARE

The following are the drawbacks present in the currently used steganography software:-

1. Reduction in number of colors in the cover image to keep the total number of unique colors less than 256.
2. Restrictions on the image size in terms of the number of pixels e.g. image size must be 320x200 pixels. If the size is less than the prescribed values then black color is padded to the image. If the size is greater than the prescribed value then the image is cropped to fit in.
3. Restrictions on the type of image, e.g., 256 bitmaps can only be used.
4. Restrictions on the dimensions of the picture in terms of pixels.
5. Data is embedded serially in LSB of each byte starting from byte next to the header. Therefore the message can be easily decoded from the image when the LSB from each byte is decoded, starting from the byte next to the header. This needs the aid of a cryptographic algorithm to make the data secure.
6. Low Bit Encoding is performed serially in the LSB of the audio stream. This also needs the aid of a cryptographic algorithm to make the data secure.
7. Most of the tools allow embedding of only single files.

The above mentioned details have been clearly explained in [JJ98A].

5. PROPOSED ALGORITHMS FOR CRYPTOGRAPHY AND STEGANOGRAPHY

5.1. Cryptographic Algorithm

5.1.1. Encryption Procedure

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

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

The steps involved in the encryption procedure are as follows:

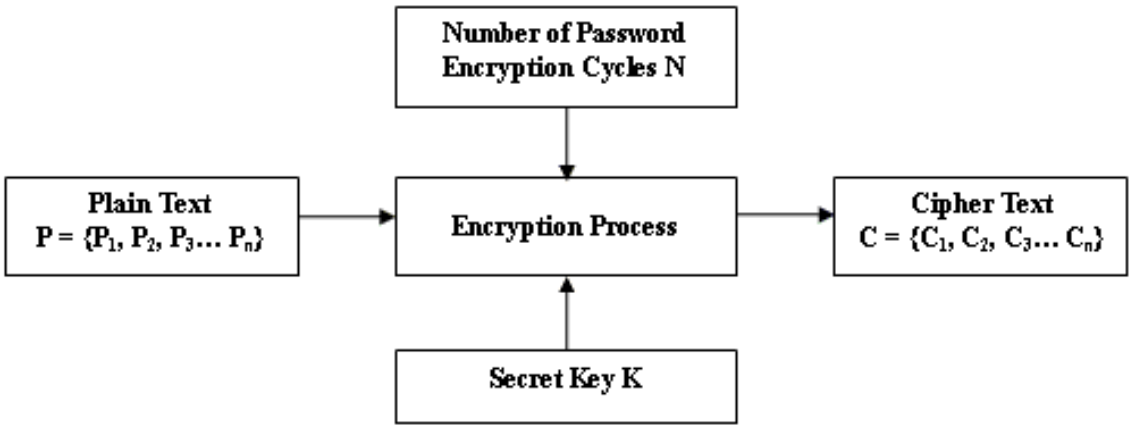


Fig. 1. Data Encryption Process

1. The key 'K' is converted into a 64 bit offset value.

2. This 64 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.

3. 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 64 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

4. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 64 bit offset.

5. The block size for file encryption is equal to key size i.e. 64 bits.

6. 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.

7. In the first stage there are various methods for generating the intermediate cipher text out of which one method is selected at random. The 64 bit offset is used for:

(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.

8. The 64 bit offset is divided into 8 bytes (8 parts). The 64 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 64 bit result. The 64 bit offset is cyclically right shifted by 8 bits and again Xor-ed with the 64 bit result obtained from the previous operation. The same operation is repeated for 8 iterations.
(ii) Each byte is Xor-ed with a byte of plain text to get a 64 bit result. The 64 bit offset is right shifted (non cyclic) by 8 bits and again Xor-ed with the 64 bit result obtained from the previous operation. The same operation is repeated for 8 iterations.

(iii) Each byte is Xor-ed with a byte of plain text to get a 64 bit result. The 64 bit offset is cyclically left shifted by 8 bits and again Xor-ed with the 64 bit result obtained from the previous operation. The same operation is repeated for 8 iterations.

(iv) Each byte is Xor-ed with a byte of plain text to get a 64 bit result. The 64 bit offset is left shifted (non cyclic) by 8 bits and again Xor-ed with the 64 bit result obtained from the previous operation. The same operation is repeated for 8 iterations.

(v) Shuffle 8 parts of the offset and Xor with the plain text to get a 64 bit result. Shuffle 8 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 8 iterations.

(vi) Shuffle 8 parts of the offset and Xor with the plain text to get a 64 bit result. Shuffle 8 parts of the offset each and every time (without reinitializing the pseudorandom generator). The same operation is repeated for 8 iterations.

(vii) Shuffle 8 parts of the offset and Xor with the plain text to get a 64 bit result. The 64 bit offset is cyclically right shifted by 8 bits and again Xor-ed with the 64 bit result obtained from the previous operation. The same operation is repeated for 8 iterations.

(viii) Shuffle 8 parts of the offset and Xor with the plain text to get a 64 bit result. The 64 bit offset is right shifted (non cyclic) by 8 bits and again Xor-ed with the 64 bit result obtained from the previous operation. The same operation is repeated for 8 iterations.

(ix) Shuffle 8 parts of the offset and Xor with the plain text to get a 64 bit result. The 64 bit offset is cyclically left shifted by 8 bits and again Xor-ed with the 64 bit result obtained from the previous operation. The same operation is repeated for 8 iterations.

(x) Shuffle 8 parts of the offset and Xor with the plain text to get a 64 bit result. The 64 bit offset is left shifted (non cyclic) by 8 bits and again Xor-ed with the 64 bit result obtained from the previous operation. The same operation is repeated for 8 iterations.

(xi) Each byte of plain text is Xor-ed with randomly selected value between 0 and 255 to get a 64 bit result. The 64 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 8 iterations.

(xii) Each byte of plain text is Xor-ed with randomly selected value between 0 and 255 to get a 64 bit result. The 64 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 8 iterations.

(xiii) Each byte of plain text is Xor-ed with randomly selected value between 0 and 255 to get a 64 bit result. The 64 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 8 iterations.

(iv) Each byte of plain text is Xor-ed with randomly selected value between 0 and 255 to get a 64 bit result. The 64 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 8 iterations.

(xv) Each byte of plain text is Xor-ed with randomly selected value between 0 and 255 to get a 64 bit result. Each byte of the 64 bit result is Xor-ed with randomly selected value between 0 and 255 (without reinitializing the pseudorandom generator). The same operation is repeated for 8 iterations.

There are many other ways to generate the intermediate cipher text. Based on the requirement of security 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.

9. 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 64 bit cipher text for the given 64 bit plain text is obtained.

10. 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 64 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 64 bits in size. The required number of bits in the key is used for the last block. In the encryption process the offset computation process is such that the number of bits in the key i.e. the key length can be increased to any value, by changing a small factor. Thus the algorithm can work with keys of any length and as the key length increases security also increases. 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. Any other hashing routine can be used in place of the offset computation process to generate a 64 bit offset when a string key is given as input.

5.1.2. Decryption Procedure

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

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

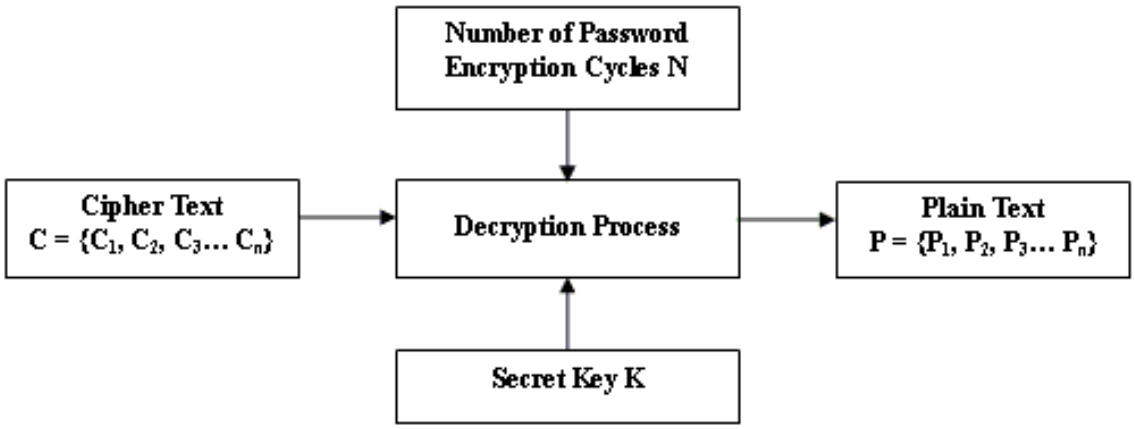


Fig. 2. Data Decryption Process

The steps involved in the decryption procedure are as follows:

1. The key 'K' is converted into a 64 bit offset value.

2. This 64 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.

3. 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 64 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

4. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 64 bit offset.

5. 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., 64 bits.

6. 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.

7. The first block of cipher text is read from the file. The 64 bit offset obtained from the encrypted cryptokey 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.

8. The next step is the conversion of intermediate cipher text to plain text. The conversion requires the computation of all the 64 bit offsets used for 8 iterations in advance, because the Xor operation between the intermediate cipher text and the 64 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 64 bit plain text block is obtained from the 64 bit cipher text block.

9. 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 64 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 64 bits then required number of bits in the key are used. Thus the entire file is decrypted.

5.2. Steganographic Algorithms

There are two main methods in the steganographic algorithms proposed. They are **Data embedding and extraction method** and the **Log file generation and tracing method**. They are explained in detail.

5.2.1. Data embedding and extraction method

The data embedding and extraction method describes how multiple files of any format are hidden inside an image or an audio stream. The inputs for embedding data are cover media, the files that are to be embedded inside the cover media, stegokey and the number of encryption cycles for stegokey encryption. The output is stego media. The block diagram of the data embedding procedure is depicted below.

The inputs for the data extraction procedure are stego media, stegokey and the number of encryption cycles for stegokey encryption. The output is set of files present inside the stego media. The block diagram of the data embedding procedure is depicted below.

5.2.1.1. Data embedding and extraction method with images as cover media

(a) Data embedding procedure

The cover media can be a JPEG, GIF or bitmap file, it is converted into a 24-bit true color bitmap image. The stego media obtained is bitmap.

The set of files ‘F’ are embedded inside cover media ‘C’ using the key ‘K’ and the number of encryption cycles for the stegokey encryption ‘N’ there by producing the stego media ‘S’ as output.

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

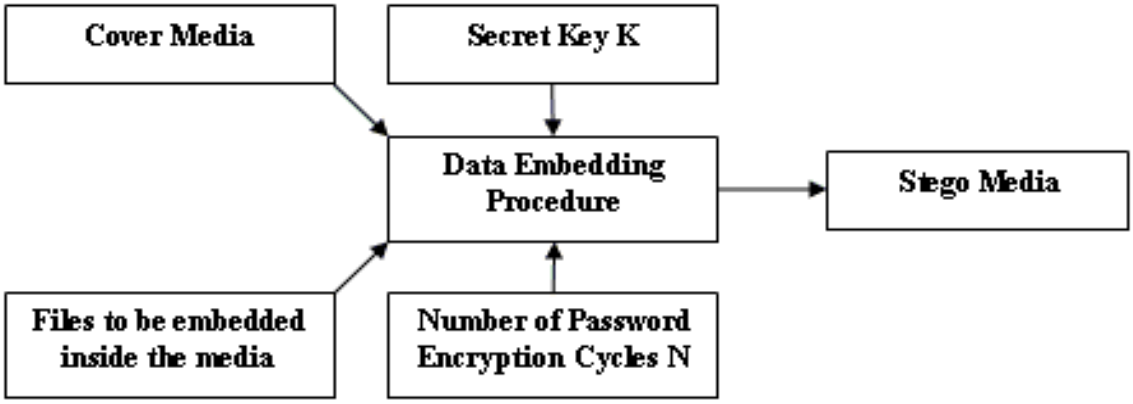


Fig. 3. Data Embedding Procedure

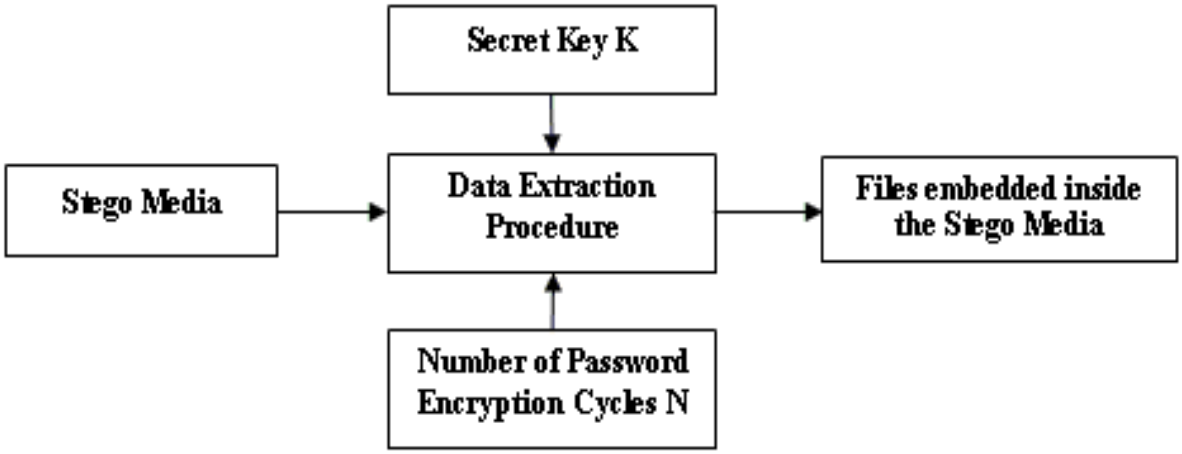


Fig. 4. Data Extraction Procedure

1. The key 'K' is converted into a 64 bit offset value.

2. This 64 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.

3. 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 64 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

4. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 64 bit offset.

5. The block size is 64 bits. The first block of data to be embedded is taken and embedded inside the cover media as follows:

(i) Every block of data is embedded byte by byte.

(ii) The data is stored by using either the least significant bit (LSB) or by using the two LSBs of the color component.

(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' and the color components present in the pixel in terms of red, green and blue are obtained.

(v) Then 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.

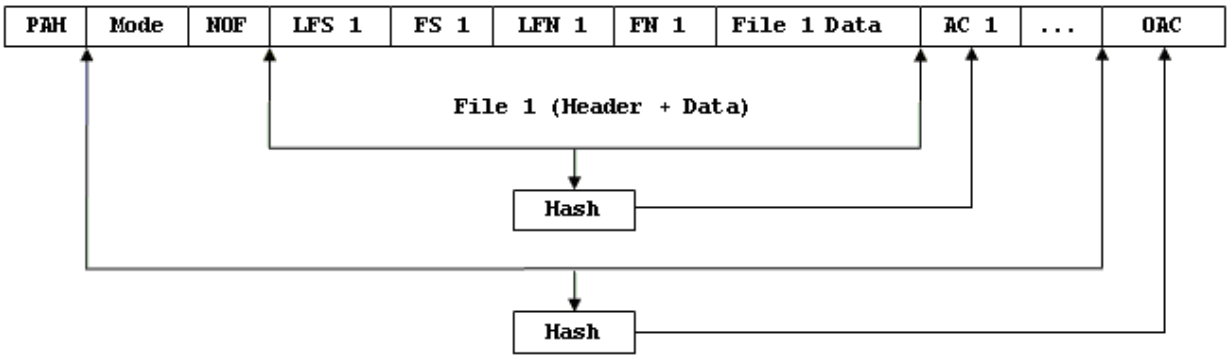
(vi) When a row, column and color component are selected at random the algorithm must ensure that the selected values are new since reusing the already used row, column and color component damages the already stored data.

(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 LSB, else the algorithm finds an unused position.

(viii) Similarly all the bytes are embedded inside the image.

6. The next block of data to be embedded is obtained. The encrypted key used for the previous block is again encrypted and the 64 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 64 bits then the last block has less than 64 bits. Thus the entire data is split into blocks and embedded into the image.

Multiple files of any format are embedded inside the image using the protocol format described below:



PAH - Password Authentication Header
Mode - Number of LSB used
NOF - Number of Files
LFS n - Length of file size (number of digits) of file n
FS n - File size n
LFN n - Length of filename (number of characters) of file n
FN n - Filename n
File n Data - Data present in file n
AC n - Authentication Checksum for file n
OAC - Overall Authentication Checksum

(Where 1 <= 'n' <= NOF)

Fig. 5. Protocol Format for Data Embedding and Extraction in Images

The **PAH** – *Password Authentication Header* ensures that the same password is used during embedding and extracting data. If wrong password is used then data cannot be extracted.

The **Mode** contains the information regarding the number of LSB used for storing data. 8 bits are used to represent the *Mode*, in future if new modes are added they can be easily represented.

The **NOF** – *Number of Files* contains the information regarding the number of files stored inside the image. 8 bits are used for representing the number of files therefore 255 files can be stored.

After embedding these details each and every file's data is embedded along with particular header information to identify the size and format of the file.

The **LFS n** – *Length of File Size of file n* represents the number of digits present in the file size. If the file size is 12000 B the length is 5. 8 bits are used for representing the length of file size.

The **FS n** – *File Size n* represents the file size of n^{th} file. The file size is converted into a string and stored. If the file size is 12000 B it cannot be represented in a single byte, and the data is embedded byte by byte. In this representation the *LFS n* denotes the length of *FS n,* since the value of *LFS n* can be up to 255 the file size can be up to 255 digits in length, which is more than enough.

The **LFN n** – *Length of Filename of file n* represents the number of characters present in the filename. If the filename is 'abc.txt' then the length is 7. 8 bits are used for representing the length of filename.

The **FN n** – *Filename n* represents the filename along with the extension. Files of various formats have different headers. To reduce the complication in processing the header format of each and every file the extension information is stored along with the filename.

The **File n Data** – *Data present in file n* represents the file data.

For each and every file the *LFS n, FS n, LFN n, FN n, File n Data* values are computed and embedded inside the image.

The **AC n** – *Authentication Checksum for file n* is the checksum computed using a hashing procedure. The *AC n* is computed over *LFS n, FS n, LFN n, FN n, File n Data*. This ensures the data integrity of the file.

The **OAC** – *Overall Authentication Checksum* is the checksum computed over all the fields except the *PAH – Password Authentication Header*. This ensures overall data integrity. Thus all the files are embedded inside the cover image.

(b) Data extraction procedure

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

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

1. The key 'K' is converted into a 64 bit offset value.

2. This 64 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.

3. 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 64 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

4. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 64 bit offset.

5. The pseudorandom generator generates the same sequence of numbers when it is initialized with the same seed value.

6. The same row, column and the color component which were selected during the data embedding process will be selected now and based on value of *Mode* the data can be extracted from the LSB. Thus the first block of data is extracted. As in the embedding process a set 's' is maintained which contains the location from which data was extracted.

7. The encrypted key used for the previous block is again encrypted and the 64 bit offset value is again computed. This is used to initialize the pseudorandom generator before extracting the next block of data.

8. The entire data is extracted in a similar manner. Based on the protocol format explained earlier 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.

5.2.1.2. Data embedding and extraction method with audio stream as cover media

(a) Data embedding procedure

The cover media and stego media are of wave file format. The wave audio files are of *RIFF – Resource Interchange File Format*. The file has a header region, other than the header region the data part is the digital form of the analog signal stored as digital samples. The wave file not only contains the digital data required to produce the sound but also additional information such as sampling rate, the type of audio data, and other critical data. Wave data comes in several formats, sampling rates, channels and resolutions (bits / sample).

The set of files 'F' are embedded inside cover media 'C' using the key 'K' and the number of encryption cycles for the stegokey encryption 'N' there by producing the stego media 'S' as output.

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

1. The key 'K' is converted into a 64 bit offset value.

2. This 64 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.

3. 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 64 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

4. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 64 bit offset.

5. The block size is 64 bits. The first block of data to be embedded is taken and embedded inside the cover media as follows:

(i) Every block of data is embedded byte by byte.
(ii) Data is stored in the LSB of the randomly selected byte from the audio stream.
(iii) A byte is split into 8 parts of 1 bit each. Each and every part is stored in LSB of the audio stream byte. When the LSB of a single byte is changed the quality is not affected much.
(iv) Then a byte is selected at random and checked whether it is present in the audio file's header range. The selected byte must not be in the header range of the audio file, else a new position is selected.
(v) When a byte is selected at random the algorithm must ensure that the selected value is new since reusing the already used byte damages the already stored data.
(vi) This is handled by maintaining a set 's' that contains the already selected values. When new byte is selected at random it is compared with the values in the set. If the values are new then the data is stored in the LSB, else the algorithm finds an unused position.
(vii) Similarly all the bytes are embedded inside the audio file.

6. The next block of data to be embedded is obtained. The encrypted key used for the previous block is again encrypted and the 64 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 64 bits then the last block has less than 64 bits. Thus the entire data is split into blocks and embedded into the audio file.

Multiple files of any format are embedded inside the audio using the protocol format described below:

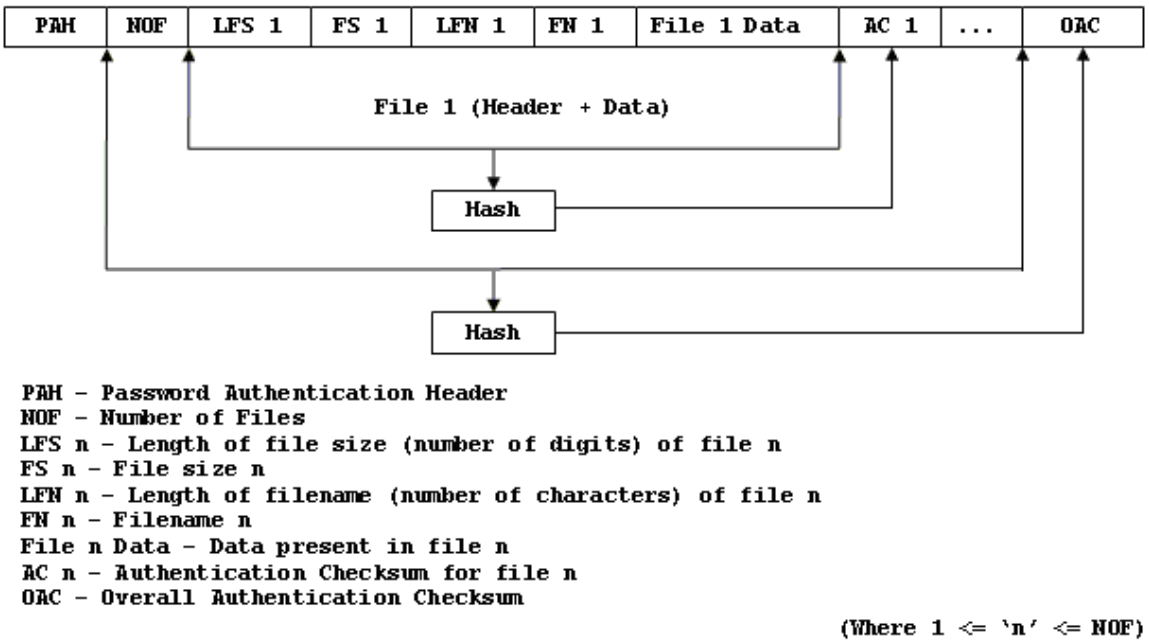


Fig. 6. Protocol Format for Data Embedding and Extraction in Audio Stream

The fields have the same meaning as in the protocol format used for images. The only difference here is that the *Mode* field is not used since only the LSB is used for storing the data. Thus all the files are embedded inside the cover audio file.

(b) Data extraction procedure

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

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

1. The key 'K' is converted into a 64 bit offset value.

2. This 64 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.

3. 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 64 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

4. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 64 bit offset.

5. The pseudorandom generator generates the same sequence of numbers when it is initialized with the same seed value.

6. The same bytes get selected and the data can be extracted from the LSB. Thus the first block of data is extracted from the LSB. Thus the first block of data is extracted. As in the embedding process a set 's' is maintained which contains the location from which data was extracted.

7. The encrypted key used for the previous block is again encrypted and the 64 bit offset value is again computed. This is used to initialize the pseudorandom generator before extracting the next block of data.

8. The entire data is extracted in a similar manner. Based on the protocol format explained earlier 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.

The functional block diagram below shows the overall operation:

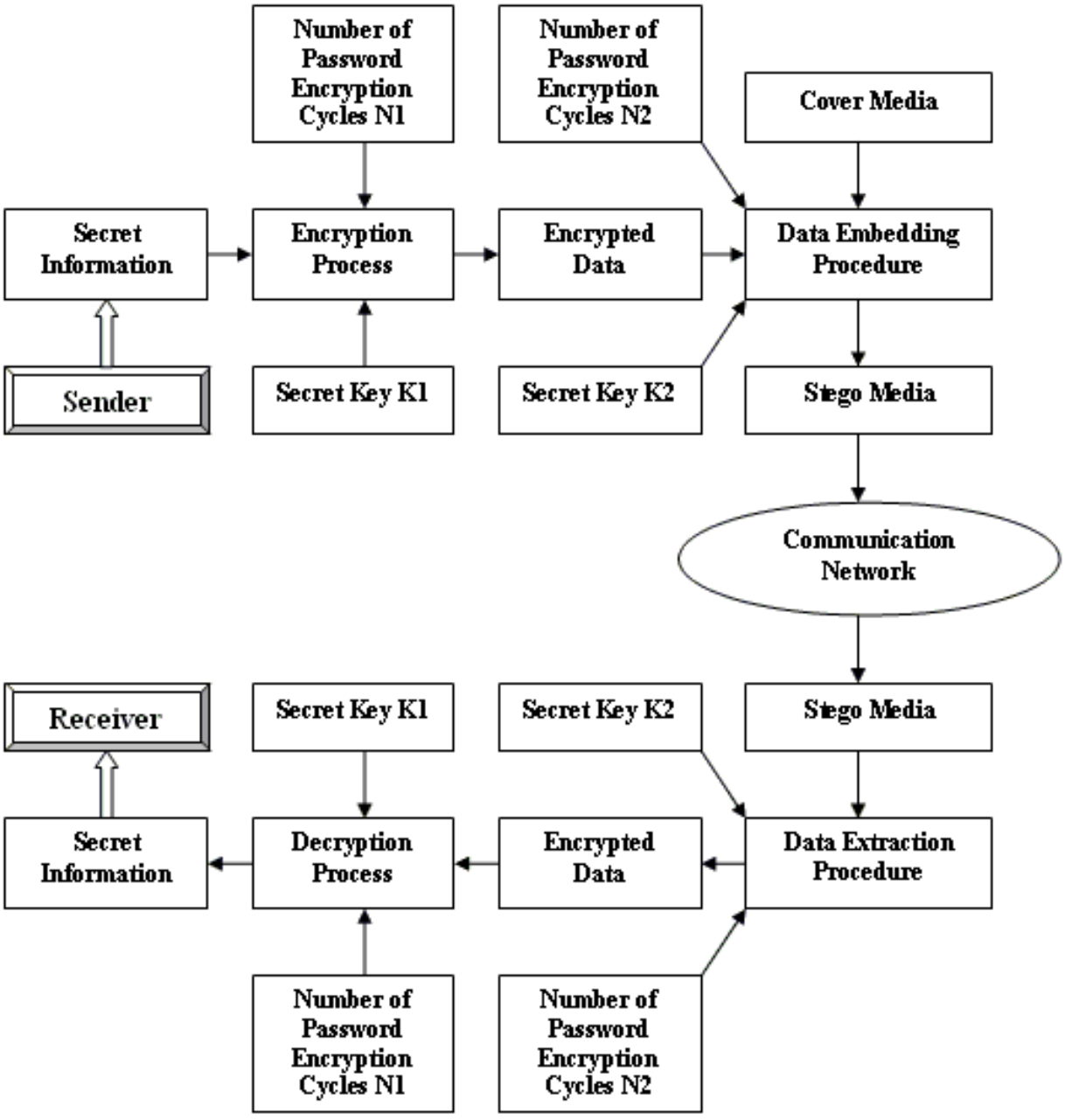


Fig. 7. Functional Block Diagram – Data Embedding and Extraction Method

The sender encrypts the secret information to be transmitted using secret key K1 (cryptokey) and number of password encryption cycles N1. The encrypted data is embedded inside the cover media using secret key K2 (stegokey) and number of password encryption cycles N2. The stego media is transmitted through the communication network and at the receiver end the data is extracted using the secret key K2 and the number of password encryption cycles N2. Then the secret key K1 and the number of password encryption cycles N1 is used to decrypt the encrypted data. The main advantage of embedding multiple files is that the information about the keys used for cryptography can be put in a separate file and sent to the receiver along with the other files where it can be extracted and the information can be used for decrypting the cipher text. Thus multiple files of any format can be embedded inside an image or an audio file.

5.2.2. Log file generation and tracing method

The log file generation and tracing method describes how multiple files of any format are encoded into log file based on an image or an audio stream or any other file as cover media. The inputs for log generation process are cover media, the files that are to be encoded into a log file, stegokey and the number of encryption cycles for stegokey encryption. The output is log file. The block diagram of the log generation procedure is depicted below.

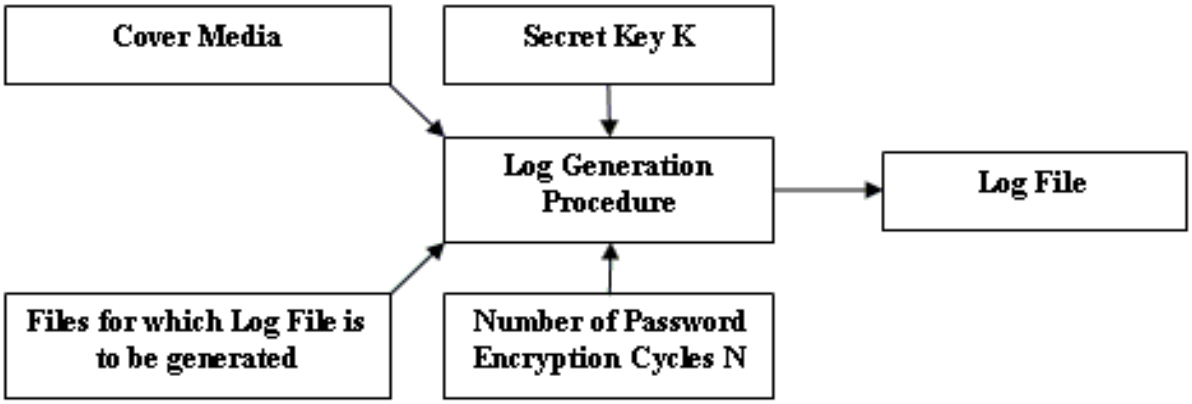


Fig. 8. Log Generation Procedure

The inputs for the log tracing procedure are log file, cover media, stegokey and the number of encryption cycles for stegokey encryption. The output is set of files which were encoded into a log file at the sender end. The block diagram of the log tracing procedure is depicted below.

5.2.2.1. Log file generation and tracing method with images as cover media

(a) Log generation procedure

The cover media can be a JPEG, GIF or bitmap file.

The set of files ‘F’ are encoded into a log file ‘L’ based on cover media ‘C’ using the key ‘K’ and the number of encryption cycles for the stegokey encryption ‘N’.

The steps involved in the log generation procedure are as follows:

1. The key ‘K’ is converted into a 64 bit offset value.

2. This 64 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.

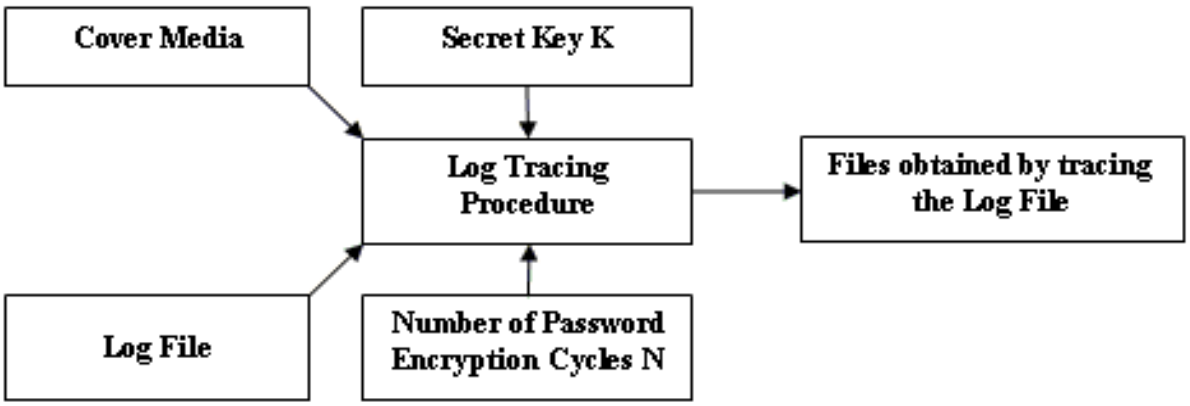


Fig. 9. Log Tracing Procedure

3. 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 64 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

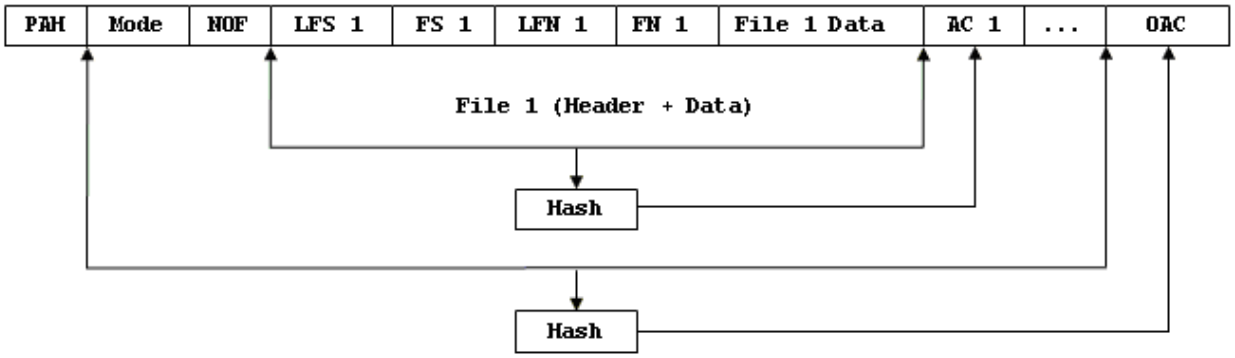
4. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 64 bit offset.

5. The block size is 64 bits. The first block of data is taken and encoded based on the cover media as follows:

(i) Every block of data is encoded byte by byte.
(ii) The log byte can be generated in two ways namely bitwise comparison process and byte wise Xor process. One of these two methods can be used or any one method can be selected at random. In the latter case the method used for log byte generation each and every block varies.
(iii) In byte wise Xor process, a pixel is selected at random by randomly selecting a row 'r' and column 'c' and the color components present in the pixel in terms of red, green and blue are obtained. A Xor operation is performed between the data byte and a randomly selected color component of the pixel to get the log byte. This is written into the log file after 64 bits of log file data is generated.
(iv) In bitwise comparison process a pixel is selected at random by randomly selecting a row 'r' and column 'c' and the color components present in the pixel in terms of red, green and blue are obtained. Then a color component is selected at random and the data bit is compared with a randomly selected bit in the color component. If the data bit and the randomly selected bit are same then the log bit is set else the log bit is reset. In a similar manner a block of log file content is generated for a block of data.
(v) In this case the set of already selected values need not be maintained since the cover media is not affected.
(vi) Thus a data block is encoded into a log file block.

6. The next block of data to be encoded is obtained. The encrypted key used for the previous block is again encrypted and the 64 bit offset value is again computed. This is used to initialize the pseudorandom generator before encoding the next block of data. If the data is not exact multiple of 64 bits then the last block has less than 64 bits. Thus the entire data is split into blocks and the log file is generated.

Multiple files of any format are encoded into a log file based on images as cover media using the protocol format described below:



PAH - Password Authentication Header
Mode - Indicates the method used for log file generation
NOF - Number of Files
LFS n - Length of file size (number of digits) of file n
FS n - File size n
LFN n - Length of filename (number of characters) of file n
FN n - Filename n
File n Data - Data present in file n
AC n - Authentication Checksum for file n
OAC - Overall Authentication Checksum

(Where 1 <= 'n' <= NOF)

Fig. 10. Protocol Format for Log File Generation and Tracing using Images as Cover Media

The fields have the same meaning as in the protocol format used for embedding data inside images. The only difference in this case is that the *Mode* indicates whether bitwise comparison or byte wise Xor method is used for log byte generation.

(b) Log tracing procedure

The set of files 'F' are decoded from the log file 'L' by comparing the log file and the cover media 'C' using the key 'K' and the number of encryption cycles for the stegokey encryption 'N'.

The steps involved in the log tracing procedure are as follows:

1. The key 'K' is converted into a 64 bit offset value.

2. This 64 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.

3. 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 64 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

4. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 64 bit offset.

5. The pseudorandom generator generates the same sequence of numbers when it is initialized with the same seed value.

6. The same row, column and the color component which were selected during the log generation process will be selected now and based on value of *Mode* the data can be decoded by comparing the log bit and the randomly selected bit of the color component. If the log bit and the randomly

selected bit are same then the data bit is set else the data bit is reset. In a similar manner a block of data is decoded from a block of log file.

7. The encrypted key used for the previous block is again encrypted and the 64 bit offset value is again computed. This is used to initialize the pseudorandom generator before decoding the next block of data.

8. The entire data is decoded in a similar manner. Based on the protocol format explained earlier 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.

5.2.2.2. Log file generation and tracing method with audio files or file any other format as cover media

(a) Log generation procedure

The cover media can be an audio file or a file of any other format.

The set of files 'F' are encoded into a log file 'L' based on cover media 'C' using the key 'K' and the number of encryption cycles for the stegokey encryption 'N'.

The steps involved in the log generation procedure are as follows:

1. The key 'K' is converted into a 64 bit offset value.

2. This 64 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.

3. 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 64 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

4. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 64 bit offset.

5. The block size is 64 bits. The first block of data is taken and encoded based on the cover media as follows:

(i) Every block of data is encoded byte by byte.
(ii) The log byte can be generated in two ways namely bitwise comparison process and byte wise Xor process. One of these two methods can be used or any one method can be selected at random. In the latter case the method used for log byte generation each and every block varies.
(iii) In byte wise Xor process, a byte is selected at random and a Xor operation is performed between the data byte and a randomly selected byte to get the log byte. This is written into the log file after 64 bits of log file data is generated.
(iv) In bitwise comparison process a byte is selected at random. Then the data bit is compared with a randomly selected bit in the selected byte. If the data bit and the randomly selected bit are same then the log bit is set else the log bit is reset. In a similar manner a block of log file content is generated for a block of data.
(v) In this case the set of already selected values need not be maintained since the cover media is not affected.
(vi) Thus a data block is encoded into a log file block.

6. The next block of data to be encoded is obtained. The encrypted key used for the previous block is again encrypted and the 64 bit offset value is again computed. This is used to initialize the pseudorandom generator before encoding the next block of data. If the data is not exact multiple of 64 bits then the last block has less than 64 bits. Thus the entire data is split into blocks and the log file is generated.

Multiple files of any format are encoded into a log file based on any file as cover media using the same protocol format described earlier. The fields have the same meaning as in the protocol format used for encoding data based on images as cover media.

(b) Log tracing procedure

The set of files 'F' are decoded from the log file 'L' by comparing the log file and the cover media 'C' using the key 'K' and the number of encryption cycles for the stegokey encryption 'N'.

The steps involved in the log tracing procedure are as follows:

1. The key 'K' is converted into a 64 bit offset value.

2. This 64 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.

3. 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 64 bit offset and used to initialize the pseudorandom generator for the next cycle of key encryption.

4. After the specified number of encryption cycles the encrypted key generated in the final cycle is converted into a 64 bit offset.

5. The pseudorandom generator generates the same sequence of numbers when it is initialized with the same seed value.

6. The same byte which was selected during the log generation process will be selected now and based on value of *Mode* the data can be decoded by comparing the log bit and the randomly selected bit of the selected byte. If the log bit and the randomly selected bit are same then the data bit is set else the data bit is reset. In a similar manner a block of data is decoded from a block of log file.

7. The encrypted key used for the previous block is again encrypted and the 64 bit offset value is again computed. This is used to initialize the pseudorandom generator before decoding the next block of data.

8. The entire data is decoded in a similar manner. Based on the protocol format explained earlier 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.

The functional block diagram below shows the overall operation.

The sender encrypts the secret information to be transmitted using secret key K1 (cryptokey) and number of password encryption cycles N1. The encrypted data is encoded into log file based on the cover media using secret key K2 (stegokey) and number of password encryption cycles N2. The log file is encrypted and transmitted through the communication network and at the receiver end the log file is decrypted and then the data is decoded using the secret key K2 and the number of password encryption cycles N2. Then the secret key K1 and the number of password encryption cycles N1 is used to decrypt the encrypted data. The main advantage of encoding multiple files into a log file is that the information about the keys used for cryptography can be put in a separate file and sent to the receiver along with the other files where it can be decoded and the information can be used for decrypting the cipher text. Thus multiple files of any format can be converted into a log file based on an image file or an audio file or a file of any format as cover media. In this case any file can be used as the cover media, the hacker requires the cover media and the log file to hack the secret information being transmitted and the files that are encoded into a log file can larger than the cover media. In case of embedding data the files must be smaller than the cover media. Thus files can be secretly transmitted using the log file generation and tracing method.

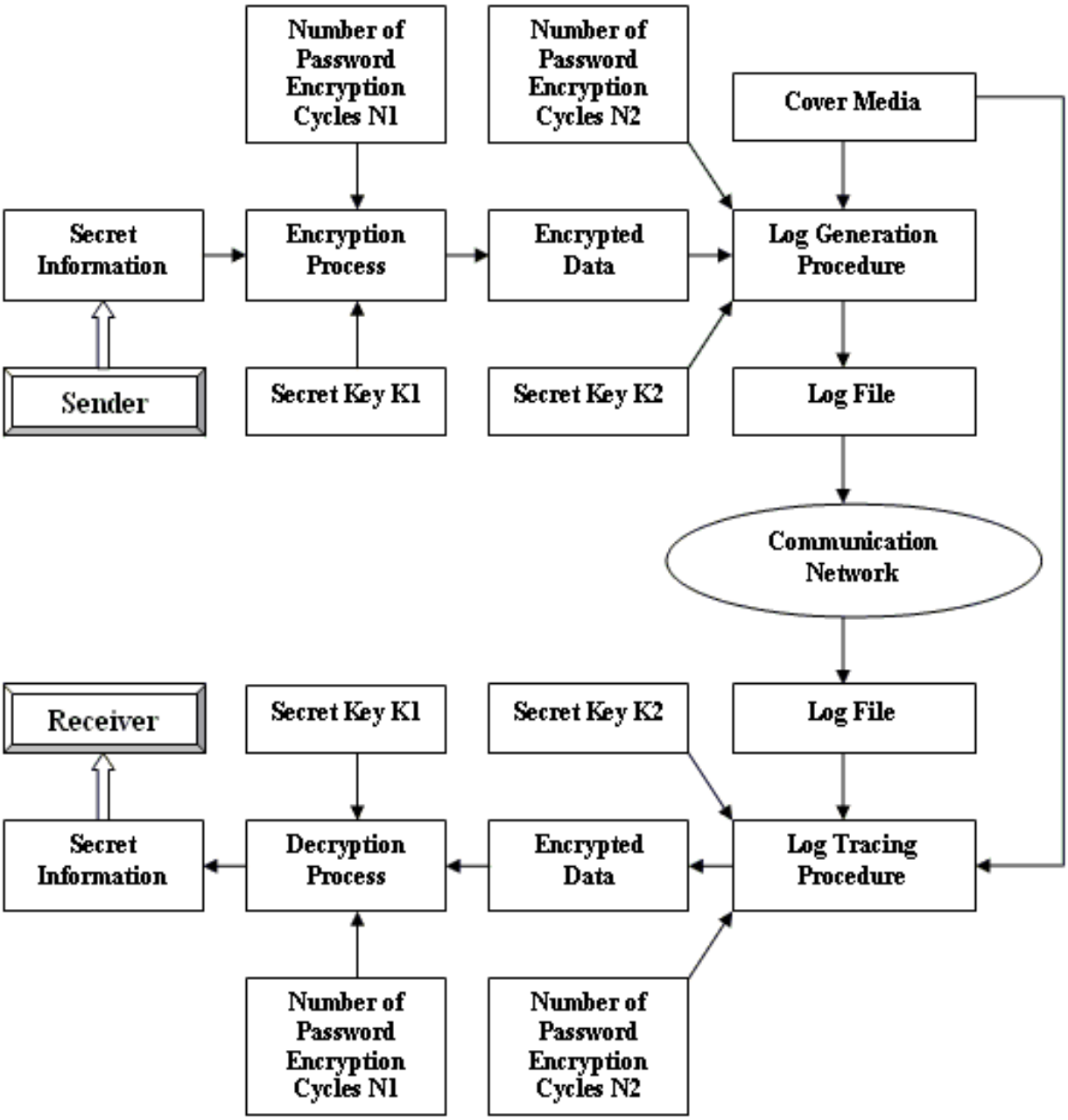


Fig. 11. Functional Block Diagram – Log File Generation and Tracing Method

6. CONCLUSION

The set of drawbacks available in the currently used procedures have been identified and solution has been provided for the existing problems. The drawbacks have been rectified with the design of new algorithms for Cryptography and Steganography. Furthermore an innovative method of data encoding without modifying the cover media has been designed. One of the major achievements is that the Cryptographic and the Steganographic algorithms have been designed such that they handle multiple files of any format. Moreover authentication is also provided for data that is being

transmitted. The new method provides acceptable quality of the stego media with no or very little distortion.

7. FUTURE COURSE OF ACTION

Currently the enhancement and of the Cryptographic and Steganographic algorithms and its implementation are in progress. The following have been identified as the future scope:

1. Implementation of the cryptographic algorithm in hardware.
2. Development of public key algorithms for cryptography and steganography.
3. Frequency domain steganography.
4. Steganography using video as cover media.
5. Error correction using various error correction techniques or development of new techniques.
6. Data compression using existing techniques or development of new techniques.
7. Development of Key Distribution Center (KDC) for cryptographic algorithm.
8. Development of VPN.
9. Development of a Random Number Generator.
10. Development of new hashing techniques.

At present the algorithms have been implemented as a software package for Windows operating system. The implementation of these algorithms for Linux operating system is in planning stage.

8. 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 Ramasubramanian P.G., Venkatakrishnan R., Rafiq M., and Venkataramanan K. for their useful discussions.

References

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

[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)