**ENCRYPTION ALGORITHM**

1: The user is asked to enter a string which he wants to embed.

2: Then the string is taken in an array which can be adjusted to suit a length that we need.

3: Then a specific number of memory locations are allocated to store the ASCII representation of each character in the string.

4: The string is converted to a stream of binary values according to each character’s ASCII code and then stored in the allocated memory sequentially.

5: User is asked to enter a stego key which is used as the seed of the pseudo-random number generator.

6: Then a particular image file is opened in binary mode with read-write access.

7: We have taken two basic structures which represents a standard 24-bit bitmap file’s FILEHEADER and INFOHEADER.

8: Both are read from the opened bitmap file and from the INFOHEADER we get information about the height and width of the image which is required to calculate the number of valid pixels.

9: According to the number of pixels we calculate the number of bits required to represent the number of pixel in the image in binary to find the largest length of text string that can be embedded.

10: We calculate the number of junk-bytes that might be present in each row of some particular bitmap image.

11: Now we allocate memory location to store an array of random positions where we will store both the length of binary stream and the actual binary bits which contain text information.

12: According to the random positions generated by the random number generator with the specific seed given by user as stego key we move the file pointer to the exact location avoiding the junk bytes

13: Then we access the pixel and modify its LSB according to the encoded text bit in binary.

**DECRYPTION ALGORITHM**

1: The user is asked to enter the stego key which is used for encryption.

2: Reading the INFOHEADER from that particular opened bitmap file in read mode we acquire information about its height and width.

3: From the height and width we calculate the number of bits to represent the largest text and we get to know about the length of the actual text string that is embedded.

4: Stego key is again used as the seed of the random number generator.

5: Random number generator generates a sequence of random number which is stored in an array.

6: According to the random positions stored in the array the file pointer is advanced to the desired location avoiding the junk-bytes.

7: We read the pixel and the LSB is obtained from the pixel which actually contain one bit of our embedded text.

8: And with 7 bits taken as a group we retrieve the actual alphabet according to its ASCII representation.

9: Proceeding as the above we develop the entire string and display it.

**STEGANOGRAPHY IN BITMAP IMAGE**

**Prepared by**

*Amrik Chatterjee*

*Chitradeep Duttaroy*

*&*

*Neel Choudhury*

Abstract

Steganography is the act of covert communications. It is referred to as the dark cousin of Cryptography. In cryptography main aim is to maintain security, whereas in steganography the main aim is to maintain secrecy. Steganography is the art of hiding secret message within benign-looking communications

known as cover texts or cover works which include picture, audio file, text file, or any video file etc. One question that any network has to give a protection against is from intruders who might sniff the data that is sent via an unprotected channel. With the help of steganography we can assure this as the cover work seems innocuous.

We have chosen a 24-bit bitmap image as the cover work in our project. Our objective is to change the LSB of each pixel according to the bits of the hidden text message. There exists three general ways to do this;

1: Sequential: A sequential selection rule embeds the message bits in individual elements of the cover Work in a sequential manner.

2: (pseudo) Random: A pseudo-random selection rule embeds the message bits in a pseudo randomly

selected subset of the cover Work. The message bits are then embedded into the elements constituting this walk. Pseudo-random selection rules typically offer better security than sequential rules. Our approach is this.

3: Adaptive: An adaptive selection rule embeds the message bits at locations that are determined based on the content of the cover work. The motivation for this is that statistical detectability is likely to depend on the content of the cover work as well. This would be our next approach.