DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DETECTING – EMBEDDING DATA IN IMAGE FILES

by
Özgür DEMİRCİ

February, 2010
İZMİR

DETECTING - EMBEDDING DATA IN IMAGE FILES

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering, Computer Engineering Program

by

Özgür DEMİRCİ

February, 2010
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"DETECTING - EMBEDDING DATA IN IMAGE FILES"** completed by **ÖZGÜR DEMİRCİ** under supervision of **INSTRUCTOR DR. MALİK KEMAL ŞİŞ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Instructor Dr. M. Kemal ŞİŞ

Supervisor

(Jury Member)

(Jury Member)

Prof.Dr. Mustafa SABUNCU
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

dedicated to my grandmother…

I would like to thank to my thesis advisor Instructor Dr. M. Kemal ŞİŞ for his help, suggestions and guidance.

I would also thank to my homemates and other collegues who supported and encouraged me all the time.

Finally I would like to thank to my mother and my father who contuniously supported and loved me in a great patience.

Özgür DEMİRCİ

DETECTING - EMBEDDING DATA IN IMAGE FILES

ABSTRACT

Steganography is a kind of information hiding technique that hide a data (text, sound, image…) in an appropriate carrier for example an image or an audio file. The carrier can then be sent to a receiver without anyone else knowing that it contains a hidden message. The objective is to hide the existence of the message and make it diffucult to read with some cryptographic methods. This study gives a new approach to BMP Steganography algorithm which is based on Least Significant Bit (LSB) insertion technique. Due to security option requirements some extra steps are added to LSB technique. This study accepts BMP images as a cover file and any message can be embedded into the cover file without significant changes perceived by a naked eye. Both encoding and decoding can be done using the visual interface of the tool.

Keywords: Stego, LSB, Least Significant Bit, Steganography, Security, Information Hiding, Image File, Cryptography

GÖRÜNTÜ DOSYALARINDA GÖMÜLÜ DATA BULMA – GÖMME

ÖZ

Steganografi bir çeşit bilgi saklama tekniğidir ve veriyi (metin, ses, görsel…) resim, ses, ya da metin dosyası gibi taşıyıcı ortamlardan kendisine uygun olanının içerisine saklamaya çalışır. Taşıyıcı dosya bir alıcıya, üçüncü şahısların içerisinde gizlenmiş bir mesaj olduğu bilgisinden habersiz olarak gönderilebilir. Amaç mesajın gizliliği sağlamak ve gizliliğinin ortadan kalktığı durumlarda ise bazı kriptografik yöntemler sayesinde üçüncü şahıslar tarafından okunabilirliğini ortadan kaldırmaktır. Bu çalışma en az öneme sahip bitleri(LSB) ekleme tekniğine dayanan BMP steganografi algoritmasına yeni bir yaklaşım sunar. Güvenlik ihtiyacından dolayı LSB tekniğine birtakım yeni adımlar eklenildi. Bu çalışma BMP resim dosyalarına çıplak gözle farkedilmeyecek şekilde başka bir veriyi saklar. Bilgi saklama ve saklanan bilgiyi geri alma işlemleri sunulan görsel arayüz ile yapılabilir.

Anahtar sözcükler: Steganografi, LSB, BMP, Bilgi Saklama, Güvenlik, Kriptografi

CONTENTS

Page

CHAPTER ONE
INTRODUCTION

1.1 Introduction

Today technology is in all parts of our lives. 20th and 21th centuries came with new inventions and some important revolutions. Especially for the last three decades there are many important things are found in the name of technology. Information is the main actor in this scene. All of the new ideas come to reality by the information. Today information investments take the major part of the companies. Because the people, the communities or the nations which manage the information gives direction to humanity. Importance of the information increases day by day. Thus the need for keeping up and using the information in secure will increase.

Information security is important for the societies such as military, banks, and companies. Boston Globe announced that the flow of 2nd world war has changed after deciphering a text (Sağıroğlu & Tunçkanat, 2002). This means, that the importance of the security of information is such as to change the flow of the history. There have been many solutions to this problem, the most widely used and investigated being Cryptography, which comes from the Greek, means secret word.

Cryptography is used to generate unreadable encrypted messages (ciphered text) by using a secret key. Cryptanalysis is used to decipher the ciphered text to get the original message back by using the secret key. But this approach is not efficient to convey the message in secure. The ciphered text is unreadable and the attacker can suspect and attack the system to get the original message.

Simmons's 'Prisoners' Problem' (Simmons, 1984) best illustrates why cryptography is not efficient. Alice and Bob have been placed in a jail guarded by a warden named Wendy, and they are allowed to communicate via Wendy. They are planning an escape, and Alice is digging out a tunnel. If Alice sends Bob an encrypted message about the progress of the escape plan: *IM BEHIND* →

ORRETBBQ, Wendy will easily suspect the message 'ORRETBBQ', and thwarts the escape plans by transferring both prisoners to high-security cells. Therefore, Alice and Bob require a new approach to camouflage the escape plan. So that, not only information security, but also information hiding becomes important. Until recently, academics and industry had given less attention to information hiding techniques than cryptology but this is changing rapidly because of the reason mentioned above.

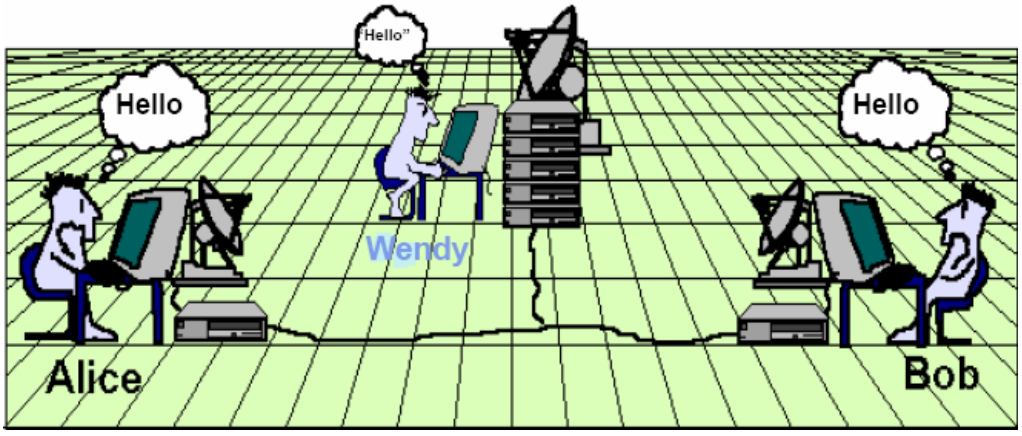


Figure 1.1 Simmons's 'Prisoners' Problem' (Simmons, 1984)

The term "Information Hiding" relates to watermarking, fingerprinting and steganography (Petitcolas, Anderson & Kuhn, 1999). Watermarking is the process of embedding marks, labels or copyright information in a data object allowing them to keep a check on piracy and only deals with the robustness of the watermark. Fingerprinting is also known hidden serial numbers and aims to distinguish an object from the similar ones. These two together are intended to prevent piracy. On the other hand, the aim of steganography is to hide messages inside other "harmless" messages in a way that does not allow any "enemy" to even detect that there is a second secret message present. This means, invisibility is the main property of the steganography; hence modifications to the cover medium may destroy the message. Steganography becomes popular after some recent news claims that terrorists have been used steganography to hide the communication in September 11 attacks (Provos & Honeyman, 2002).

Alice and Bob story can be considered from the point view of steganographic solution. If Alice and Bob had been using steganography instead of cryptography, they would have concealed the existence of a secret message. If Alice sends Bob the message, ‘it may be extremely hard interpreting nonsensical digits’ and if Bob takes the first letter of each word, he receives the Alice’s secret message ‘IM BEHIND’ without arousing Wendy’s suspicion.

Finally, it can be said that, steganography prevents the discovery of existence of a communication so that it keeps the information in secure without causing any suspect and provides a secure channel while conveying a secret message. On the other hand, cryptography prevents an unauthorized person from discovering the content of communication without knowing key. Applying steganography to an encrypted message, by combining with cryptography and steganography, provides more secure system. If an adversary detects the message that is hidden, he or she only gets the encrypted one. Then the cryptographic attacks must be applied to get the original message back.

In the following sections a detailed look in cryptography, Steganography, some detection techniques of Steganography, usage area of Steganography and the offered solution and its implementation will be seen.

For the next chapter cryptographic terminology, attacks and types of cryptographic algorithm will be analyzed. Because of using cryptography in steganographic systems to offer more secure hybrid solutions, it’s important to examine it deeply.

In chapter three a brief history of Steganography and different kinds of Steganographic techniques will be seen in detailed. Cons and pros can be analyzed to each others.

In chapter four techniques that can be use of detecting Steganography will be search. Which technique is the most threatening for our stegosystem we will decide?

In chapter five usage area of Steganography will be look into. Some examples in real world and terrorism will be check over.

In chapter six there will be a new approach to the LSB (Least Significant Bit) technique that will increase the security of the stego system which offered by me. The software interface will be search in this chapter that was implemented by me.

Conclusion, final decision and works that can be done in the future were put in the last chapter.

After the last chapter the references will be seen and finally in the appendices part there will be found source file of the software that was implemented by me.

CHAPTER TWO

CRYPTOGRAPHY

2.1 Introduction

"Over the last decade, there has been an accelerating increase in the accumulations and communication of digital information by government, industry, and by other organizations in the private sector" (Grabbe, 2003). This information must be kept secure while storing or transmitting in order to avoid unauthorized access. 'Information Security' has become a field in computer science because of this need. Over the centuries, many scientists deal with information security issues and try to set protocols and mechanism to overcome this problem.

"Achieving information security in an electronic society requires a vast array of technical and legal skills. There is, however, no guarantee that all of the information security objectives deemed necessary can be adequately met. The technical means is provided through cryptography" (Menezes, Oorschot & Vanstone, 1996).

Cryptography "from Greek *kryptós*, hidden, and *gráphein*, to write" (Wikimedia, 2009) is the art and science of keeping messages secure. Cryptographic algorithms can be divided into three categories: Secret Key Cryptography, Public Key Cryptography, and Hash Functions. DES, Triple-DES, Blowfish, AES-Rijndael, and IDEA are the well known algorithms that example the first category. RSA algorithm is the best illustration of second category. MD5, SHA and RIPEMD–160 algorithms are used for hashing.

Before introducing these algorithms, some terminology, attacks to cryptographic system and types of cryptographic algorithms are mentioned.

2.2 Terminology

The original data is called as plaintext. Encryption is the process of transformation of plaintext into unreadable form, which is called as cipher text. Decryption is the reverse of encryption, and tries to get a plaintext from a cipher text. The key is only known by a sender and receiver and is used to secure the plaintext. All together form a cryptographic system as depicted in Figure 2.1 (Ferguson & Schneir, 2003, p. 21).

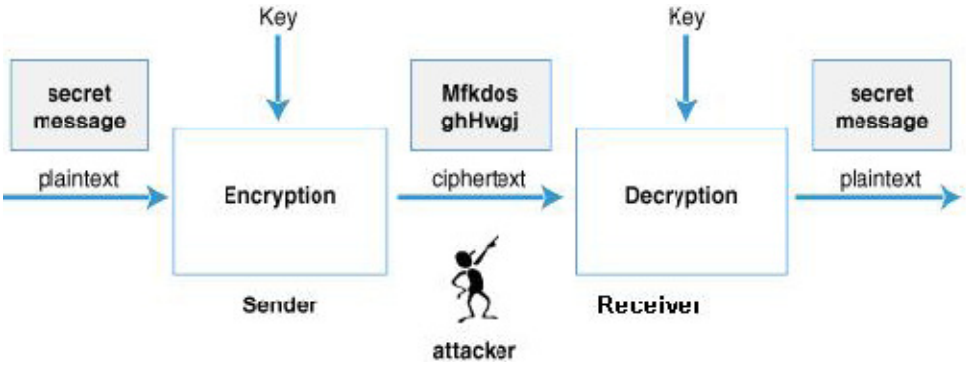


Figure 2.1 General overview of cryptography.

2.3 Attacks

Cryptographic system provides a secure communication between sender and receiver by keeping away the plaintext (or the key, or both) from attackers (also called adversaries, eavesdroppers). As a response, attackers have tried to develop some techniques which allow recovering a plaintext without having knowledge of key from a cipher text. These techniques are called as cryptanalysis.

There are two general approaches to attacking an encryption algorithm (Stallings, 2003, p.27):

1. *Cryptanalysis*: "Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext-cipher text pairs" (Stallings, 2003, p.27). There are three general types of cryptanalytic attacks.

a. *Cipher Text Only*: This is the most difficult attack type because the attacker can have only a set of cipher text which is encrypted using same encryption algorithm. But he/she does not know anything about the plaintext, key, or encryption algorithm. The attack is successful if the corresponding plaintexts can be recovered, or even better, the key.

b. *Known Plaintext*: The attacker has both plaintext and its encrypted version, cipher text, and tries to find the key.

c. *Chosen Plaintext*: The attacker has some cipher text which corresponds to some selected plaintext. If the encryption algorithm and cipher text are available, the attacker encrypts plaintext looking for matches in cipher text.

2. *Brute-Force Attack*: In a cryptographic system, the key is secret. The encryption and decryption algorithms are available to everyone. So, if the key was predicted, the cipher text would be converted to a plain text. Cryptanalysis focuses this fact. To predict key, a brute force attack, which tries every possible key, can be used. If key size becomes longer, the required time to get key goes infinite.

Table 2.1 shows the time required for different key size. The third column of table considers the results for a system that can process one key per millisecond whereas the final column is the presentation of system that can process 1 million keys per millisecond.

Table 2.1 Average time required for brute force attacks (Stallings, 2003, p.29)

Key Size (bits)	Number of alternative Keys	Time required at 1 encryption/μs	Time required at 106 encryption/μs
32	$2^{32} = 4.3 \times 10^{9}$	2^{31} μs = 35.8 min.	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	2^{55} μs = 1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	2^{127} μs = 5.4×10^{24} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	2^{167} μs = 5.9×10^{36} years	5.9×10^{30} years

2.4 Types of Cryptographic Algorithms

"Cryptographic algorithms can be classified according to the number of keys used for encryption and decryption" (Kessler, 1998). There are three types:

- *Secret Key Cryptography*: Same key is used for both encryption and decryption.
- *Public Key Cryptography*: One key is used for encryption and another for decryption.
- *Hash Functions*: A mathematical transformation is used to encrypt information, and any key may not be used.

2.4.1 Secret Key Cryptography

In secret key cryptography, a single key is used for both encryption and decryption. Because same key is used for both functions, secret key cryptography is also called as symmetric encryption. The most common secret key cryptography schemes are DES, Triple DES, AES, IDEA, and Blowfish.

"Symmetric key ciphers can be broadly grouped into block ciphers and stream

ciphers" (Wikimedia, 2009). Stream ciphers operate on a single bit at a time; a block cipher divides the input into blocks of fixed length and operates one block at a time. DES, Triple DES, AES, IDEA, and Blowfish are all block cipher.

2.4.1.1 Data Encryption Standart (DES)

"DES is the most widely used a symmetric block cipher data encryption algorithm and was developed by the National Bureau of Standards (NBS) with the help of the National Security Agency (NSA) in 1977" (Grabbe, 2003).

DES takes a 64-bit block of plaintext, and a 64-bit (in fact, only 56-bits are used) secret key for encryption and produces a 64-bit block of cipher text by applying a series of bit permutation, substitution, and recombination operations. It is also a symmetric algorithm because of using the same key for decryption.

Figure 2.2 epicts the general overview of DES algorithm. The details of each round are depicted in Figure 2.3. Binary XOR (or shortly XOR) operation is used. A binary XOR produces '1' if the inputs are different and produces '0' if both of inputs are 0 or 1.

The followings are the steps of encryption algorithm, the details of which are given in.

1. *Step 1*: *Process the Key*. 64-bit key is taken as an input. Every 8-bit is ignored for security purposes. To generate sub keys the following sub steps are used as depicted in the right hand side of Figure 2.2.

    a. Pass the 64-bit key through a permutation called Permuted Choice. The permuted key contains only 56 bits of the original key.

    b. Split the 56-bit key into two halves, each of which is 28 bits.

c. Create 16 blocks by applying left shift operations to the previous block.

d. Pass these shifted values through another permutation table.

e. 48-bit output is a sub key.

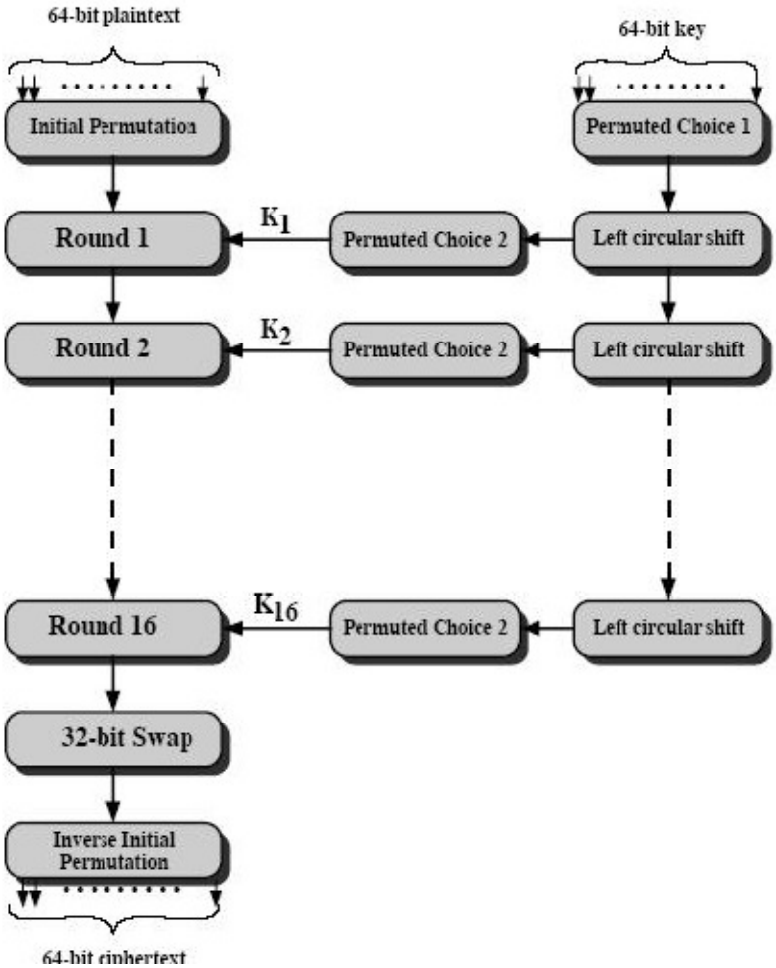


Figure 2.2 General overview of DES encryption (Stallings, 2003, p. 74)

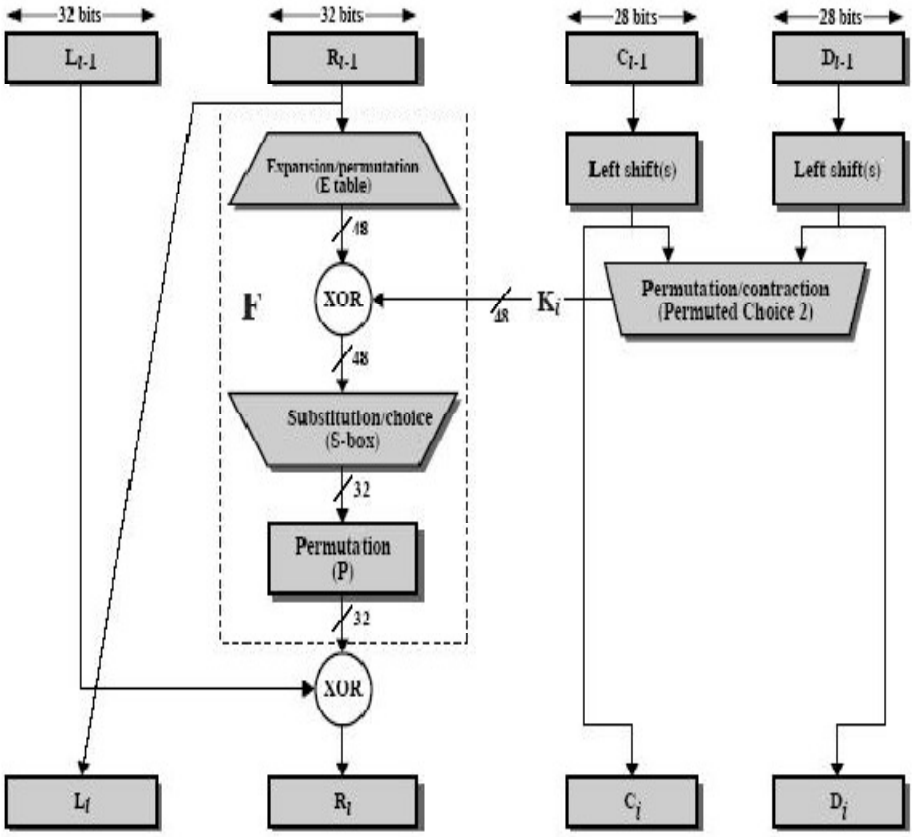


Figure 2.3 Details of F function. (Stallings, 2003, p. 77)

2. *Step 2: Process a 64-bit block of plaintext*. 64-bit data block is taken as an input. If the block is shorter than 64 bits, it will be expanded to 64 bits. To encode each block the following sub steps are used as depicted in the left hand side of Figure 2.2.

    a. Pass the 64-bit plain text through an initial permutation.

    b. Iteration count is set to 0.

    c. Split the permuted block into two halves, each of which is 32 bits.

    d. Follow these sub steps depicted in Figure 2.3 to understand how F function works.

        i. Expand the right half to 48 bits to be able to apply a binary XOR

operation with the corresponding sub key that is generated in step 1. Output of previous step is 48 bits, split these bits into 8 segments, each of which is 6 bits using substitution boxes (Sbox). The S-boxes are numbered from 1 to 8.

ii. Output of previous step is 32 bits; pass these bits through a permutation table.

e. The output of F function is XORed with left half of data and moved into right half of next step. Right half of data is moved into left half of next step without any change.

f. Increment the iteration count by 1.

g. If the iteration count is 16, apply a final permutation. Otherwise, get the next block and go to step 'c'.

The same algorithm can be used for decryption with an exception that the sub keys must be applied in reverse order.

Brute-force is a well known attack which tries every possible key in turn to get the plaintext from a cipher text. Therefore, the key length determines the feasibility of the algorithm, and the 56 bits used by DES is not sufficient to resist to this attack. "The Electronic Frontier Foundation has sponsored the development of a crypto chip named 'Deep Crack' that can process 88 billion DES keys per second and has successfully cracked 56 bit DES in less than 3 days" (Mercury, 1999). "The recent design of a $1M machine that could recover a DES key in 3.5 hours" (Schneier, 1995).

2.4.1.2 Triple DES

Triple-DES is a variant of DES. It uses two (or three) 56-bit DES keys, first is used for DES encryption, and second is for the decryption of the encrypted DES

message. Since the second key is different from first key, the decryption behaves like as an encryption. The twice-encrypted message then encrypted again with the first key (or a third key) to get the cipher text. Figure 2.4 and Figure 2.5 depicts the block diagram of encryption and decryption algorithm respectively.

Figure 2.4 Triple DES encryption algorithm

Figure 2.5 Triple DES decryption algorithm

Triple DES is more secure than DES, but it is quite slow. In addition, since DES and Triple DES use a fixed 64-bit block size, this is a drawback. Advances in cryptographic attacks has added some new security criteria and both DES and Triple DES could not meet all the needs of a secure system. Therefore, a new robust encryption algorithm was required to replace DES and Triple DES. "National Institute of Standards and Technology (NIST) asked for proposals from the cryptographic community" (Ferguson & Schneir, 2003, p. 55). Then, NIST selected an algorithm as a new encryption Standard in 2001 and this algorithm is also known by the name of Rijndael.

2.4.1.3 Advanced Encryption Standard (AES) – Rijndael

"AES is a symmetric and block cipher and was designed by Joan Daemen and Vincent Rijmen" (Wikimedia, 2009). AES takes a 128-bit block of plaintext and produces a 128-bit block of cipher text. "In most ciphers, the round transformation has the Feistel Structure. In this structure typically part of the bits of the intermediate state are simply transposed unchanged to another position" (Daemen & Rijmen, 2002). Unlike DES, AES is not a Feistel cipher.

The followings are the steps of AES encryption algorithm; the details of which are given in. Figure 2.6 shows the general overview of encryption and decryption of AES algorithm.

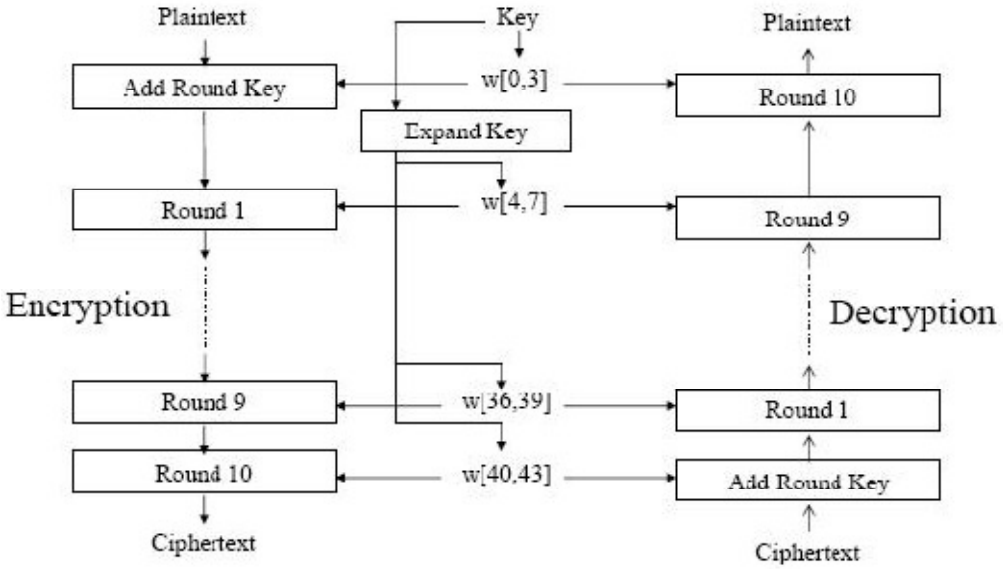


Figure 2.6 General overview of AES algorithm (Wikimedia, 2009)

1. *Step 1:* Perform a binary XOR of the 128-bit block with a sub key as shown in Figure 2.7. This operation will be named as 'Add Round Key' in the rest of chapter.

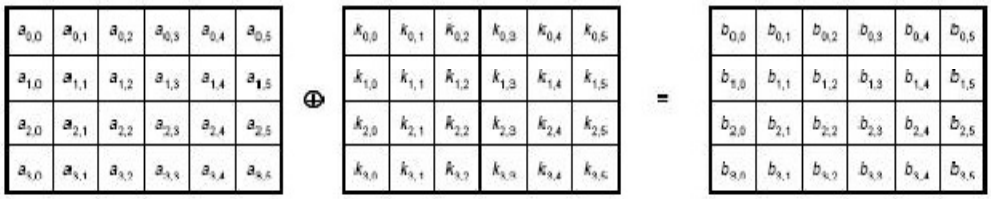


Figure 2.7 Add round key

2. *Step 2:* AES has 10–14 rounds with respect to size of key. At each round do the followings, but last round skips the mixing column operation step

    a. *Substitute Bytes:* Byte to byte substitution using S-boxes as shown in Figure 2.8.

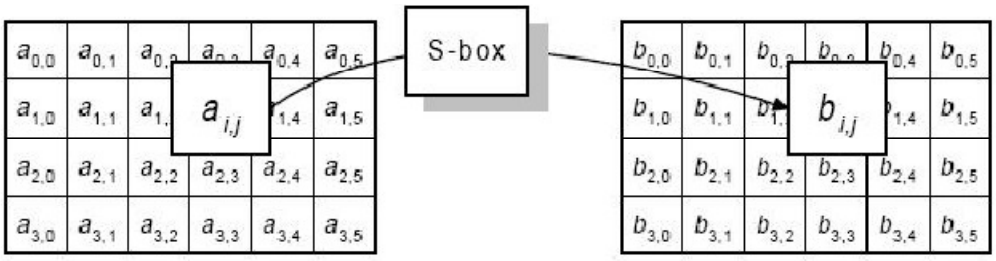


Figure 2.8 Substitution bytes

b. *Shift Rows:* Bytes are rearranged by applying cyclic shift as shown in Figure 2.9.

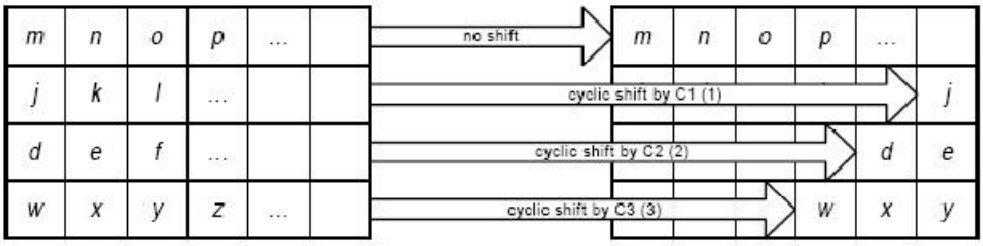


Figure 2.9 Shift rows

c. *Mix Columns:* The bytes are mixed as a group of four using a linear mixing function as shown in Figure 2.10 "Linear function means that each group output bit of the mixing function is the XOR of several of the input bits" (Ferguson & Schneir, 2003, p.56).

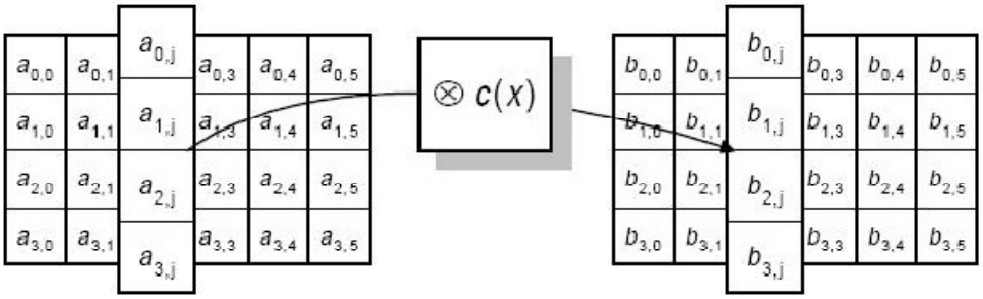


Figure 2.10 Mixing function

d. *Add Round Key:* Perform a binary XOR of the 128-bit block with an expanded key.

3. Step 3: Key expansion and round key selection operations are examined as a new

step but in fact they work parallel with step 2.

a. “Key Expansion takes as input a 16 byte key and produces a linear array of 156 bytes” (Stallings, 2003,p. 160) as shown in Figure 2.11

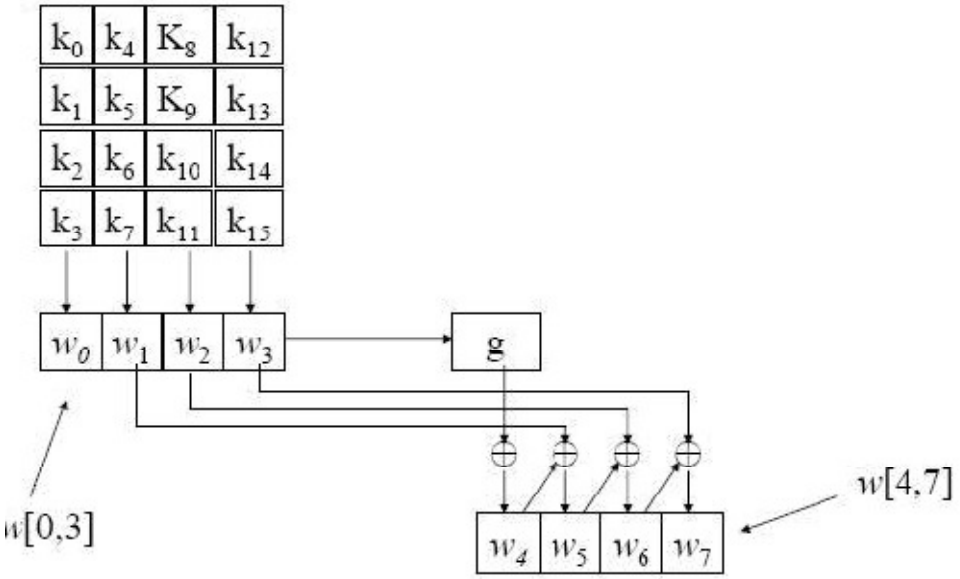


Figure 2.11 Key expansion

b. At each round, four bytes of the expanded key array is selected in an order.

AES decryption algorithm is a bit different from AES encryption algorithm. It needs the inverse of some operations: For example, matrix in the Mix Column step needs to be replaced with its inverse, and the look up tables of S-boxes in the Byte Sub step needs to be replaced with its inverse.

Before NIST has announced that DES was sufficient, cryptographic community has noticed this fact and tried to develop new encryption algorithms to substitute DES. Blowfish and IDEA are the most common ones and they are still resistant to the cryptographic and brute-force attacks.

2.4.1.4 International Data Encryption Standard (IDEA)

IDEA was developed by Xuejia Lai and James L. Massey in Switzerland in 1991 (Wordiq, 2004) to replace the DES standard and then patented by the Swiss firm of 16 Ascom. Since IDEA is used with a well known file and email protection program, Pretty Good Privacy, it is still popular.

IDEA is a symmetric block cipher and takes a 64-bit block (4 16-bit blocks) of plaintext and a 128-bit secret key (8 16-bit sub key) for encryption and produces a 64-bit block of cipher text by applying a series of modular addition and multiplication and binary XOR operations. There are 8 rounds and Figure 2.12 depicts a round. XOR, addition, and multiplication are denoted with '□', ' ', '□' respectively. K is the abbreviation of the key.

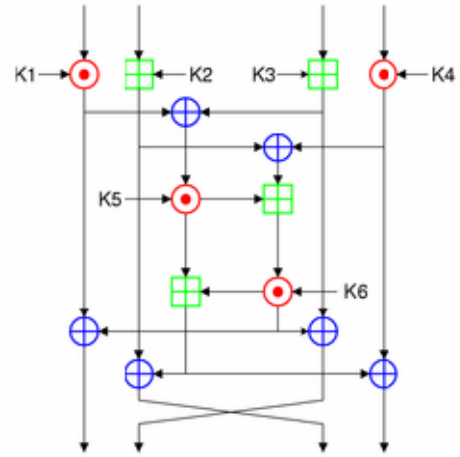


Figure 2.12 A round of IDEA (Wordiq, 2004)

The followings are the steps of encryption algorithm; the details of which are given in (Lai, 1992).

1. *Step 1:* 52 sub keys are generated. First eight sub keys are given as input. The next eight sub keys are obtained after a 25-bit circular left shift, and this is repeated until all encryption sub keys are derived.

2. *Step 2*: The following sub steps are performed 8 times.

    a. Multiply the first sub block and the first sub key.
    b. Add the second sub block and the second sub key.
    c. Add the third sub block and the third sub key.
    d. Multiply the fourth sub block and the fourth sub key.
    e. XOR the results of steps ‘a’ and ‘c’.
    f. XOR the results of step ‘b’ and ‘d’.
    g. Multiply the results of step ‘e’ with the fifth sub key
    h. Add the results of steps ‘f’ and ‘g’.
    i. Multiply the results of step ‘h’ with the sixth sub key
    j. Add the results of steps ‘g’ and ‘i’.
    k. XOR the results of steps ‘a’ and ‘i’.
    l. XOR the results of steps ‘c’ and ‘i’.
    m. XOR the results of steps ‘b’ and ‘j’.
    n. XOR the results of steps ‘d’ and ‘j’.

3. Step 3: Finally, swap ‘b’ and ‘c’. The results of steps ‘j’, ‘k’, ‘m’, and ‘n’ are the outputs of step 2 and as inputs for step 3.

    a. Multiply the first sub block and the forty-ninth sub key.
    b. Add the second sub block and the fifty sub key
    c. Add the third sub block and the fifty-first sub key
    d. Multiply the fourth sub block and the fifth-second sub key.
    e. Combine these four sub block to get the cipher text.

The same algorithm can be used for decryption with an exception that the sub keys must be applied in reverse order using different calculation. First four sub key is calculated like this: KDec(1) = 1/K(49), KDec(2) = -K(50), KDec(3) = -K(51), KDec(4) = 1/K(52). IDEA is faster and more secure than DES encryption. But IDEA is patented which restricts its commercial use.

2.4.1.5 Blowfish Algorithm

"Blowfish is a symmetric block cipher that was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms" (Tropical, 2007) The premise of this work was not to have any algorithm that was secure, unpatented and freely available at that time.

Blowfish takes a 64-bit block of plaintext, and produces a 64-bit block of cipher. Differently, Blowfish accepts a variable length key, up to 448 bits. Figure 2.13 and Figure 2.14 depicts the general overview of Blowfish algorithm.

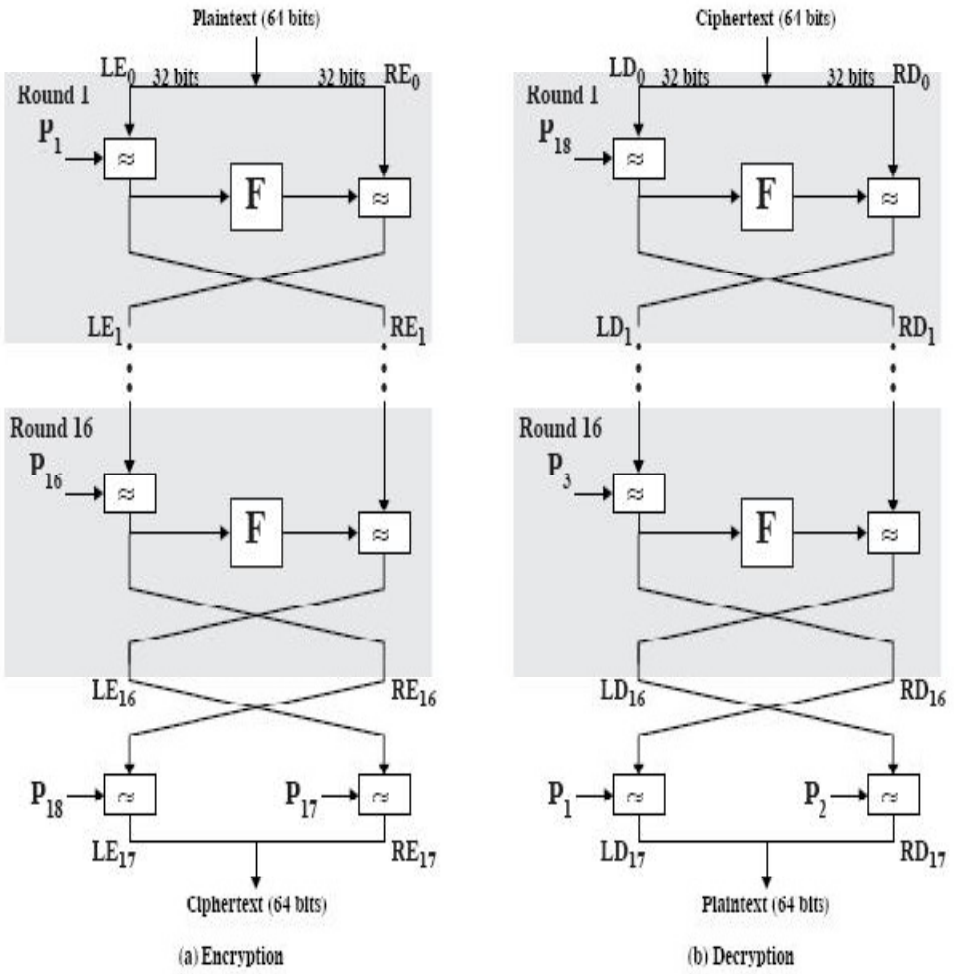


Figure 2.13 General overview of Blowfish (Gatliff, 2003).

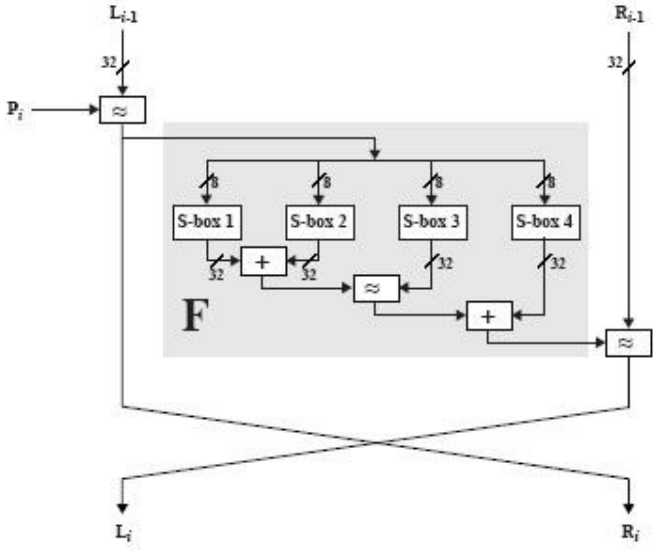


Figure 2.14 Details of F function (Gatliff, 2003)

"Blowfish has 16 rounds. Each round consists of a key-dependent permutation, and a key- and data-dependent substitution. All operations are XORs and additions on 32-bit words" (Schneier, 1995).

"The algorithms consist of two parts: a key expansion part and a data-encryption part" (Schneier, 1993). Followings are the steps of algorithm and the details of which are given in (Schneier, 1993).

1. *Key Expansion Part*: The algorithm uses P-array with 18 entries and four Sboxes (also called S array) with 256 entries while generating sub keys.

    a. Initialize P and S arrays.

    b. Split key into 32-bit blocks and apply XOR between key and the initial elements of the P and S arrays. The results are written back into the array.

    c. Encrypt the 64 bit block using the current P and S arrays, replace P [1]

and P [2] with the output of the step b.

d. Encrypt the output of step c using the current P and S arrays, replace P [3] and P [4] with the output of the step c.

e. The algorithm continues this process until all elements of P and S arrays are updated.

2. *Data Encryption Part*: Figure 2.13 depicts the encryption operation. The details of F function are shown in Figure 2.14. The Blowfish encryption algorithm works in a way such as DES do with only three exceptions. First is initial permutation which is skipped during Blowfish encryption. Second is F function which a bit more complex than DES. This makes Blowfish more resistant to cryptographic attacks. Third is sub key generation process which adds a complexity for brute force attacks.

The same algorithm can be used for decryption with an exception that the sub keys must be applied in reverse order.

2.4.2 Public Key Cryptography

In public key cryptography, two keys are used. One key is for encryption and a different but related key is for decryption. Since different keys are used, public key cryptography is also called as asymmetric encryption. The most common public key cryptography scheme is RSA.

2.4.2.1 Rivest – Shamir – Adleman (RSA)

RSA is the most widely used an asymmetric block cipher data encryption algorithm and “was developed in 1978 by Ron Rivest, Adi Shamir, and Leonard Adleman” (Ferguson & Schneir, 2003, p.223)

RSA takes a k-bits block of plaintext and each block having a binary value less than ‘n’ where $2k < n \leq 2k + 1$. RSA works as follows:

1. Take two large primes, p and q, and compute their product n = pq; n is called the modulus.
2. Choose a number, e, less than ‘n’ and relatively prime to (p-1)(q-1), which means ‘e’ and (p-1)(q-1) have no common factors except 1.
3. Find another number ‘d’ such that (ed - 1) is divisible by (p-1)(q-1). The values ‘e’ and ‘d’ are called the public and private exponents, respectively.
4. The public key is the pair (n, e); the private key is (n, d).
5. Finally, sender encrypts a message, m, to create the cipher text, c, by exponentiation: c = me mod n.
6. Decrypt the cipher text by exponentiation: m = cd mod n. Since only receiver knows ‘d’, only receiver can decrypt this message.

Since RSA uses mathematical operations, it works very slowly in comparison to secret key cryptographic algorithms. Also, the key distribution is very difficult. Because public and private key pairs are calculated from ‘p’ and ‘q’, both the sender and receiver must know ‘p’, and ‘q’. So it is not used as an encryption routine in this thesis.

2.4.3 Hash Functions

A hash function takes a message with any size as an input and then produces a fixed size result as an output. It is also called “message digest functions and the output is called as a digest or hash value” (Ferguson & Schneir, 2003, p.83). Generally, the name hash function is more common. The most common hash functions are MD5, SHA (and its variants) and RIPEMD–160. In fact, hash functions are a bit different than two methods mentioned above. A hash function not only used for encryption, but also used for authentication, simple digital signature, and pseudo random number generation. So before explaining hash functions, some terminology is given.

2.4.3.1 Basic Terminology of Hash Functions

A hash function, denoted with H, works by producing a fixed length digest, denoted with h, from the variable length message, denoted with m. Here is the general notation of hashing:

$$h = H(m)$$

There are several requirements for a hash function (Stallings, 2003, p.328).

1. *One-way Property*: For any given value *h, it* is computationally infeasible to find *x* such that $h = H(x)$.
2. *Collision resistance*: For any given block *x, it* is computationally infeasible to find $y \neq x$ with$H(\ y) = H(x)$. This is called as collision. In fact, no hash function exists with collision free property. Therefore, collision resistance means, that they cannot be found easily.

All hash functions are iterative functions and accept a sequence of n-bit blocks of input, $n\ m\ ,m\ ,m\ ,.....,m\ 1\ 2\ 3$. The block of the input is processed using some logical operations such as XOR, AND, OR, and NOT. This process starts with a fixed value, 0 *h* and continues by calculating $i\ h$ where $(\ )\ i\ i\ i\ h\ H\ h,\ m\ -1 =$ and 'i' is the range of 1 to n, and ends where the value of 'i' reaches 'n'.

Hash functions are used for authentication by adding a hash value at the end of message that are transmitted to the recipient as shown in Figure 2.15-a. The recipient performs same hash function to the message to generate a new hash value. The received hash value is compared with the calculated hash value. If they are same, the message is authenticated. If the hash code is encrypted before adding to the message, this provides a digital signature as shown in Figure 2.15-b.

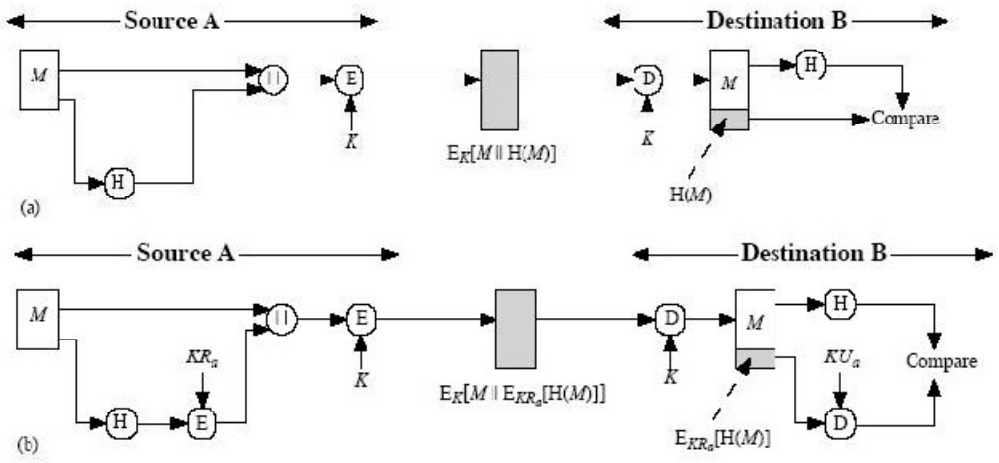


Figure 2.15 Basic uses of hash function (Stallings, 2003, p.324)

2.4.3.2 Message Digest Algorithm 5 (MD5)

MD5 is a kind of message digest algorithm (or a hash function) designed by Ron Rivest. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit. The input is processed in 512-bit blocks. The algorithm consists of the following steps and the details of which are given in (Rivest, 2003).

1. *Step 1*: *Append padding bits.* The message is padded (expanded) to get the desired length which is an integer multiple of 512 bits. The number of padding bits can be in the range of 1 to 512. Padding starts with adding a single '1' bit and continues adding necessary number of '0' bits.

2. *Step 2: Append Length*: The length of message is calculated by taking the length of original message modulo 264. Figure 2.16 shows the expanded message scheme.

L * 512 bits (length of expanded message)			
K Bits		*Padding*	*Message Length*
Original Message		100…..0	K mod 2^{64}
Y_0 (512 bits)	Y_1(512 bits)	…….	Y_L(512 bits)

Figure 2.16 The expanded message looks like after applying step 1 and 2.

3. *Step 3: Initialize Message Digest buffer with hexadecimal numbers*. A 128-bit buffer contains four 32-bit registers called A, B, C, and D in which these values are stored in little-endian format, which is the least significant byte of 32-bit in the low-address byte position.

4. *Step 4: Process message in 512-bits blocks*. The MD5 function has four rounds of processing of 16 steps each, shown in Figure 2.17 a single step of MD5 processing.

Rounds have a similar structure except using different logical functions. Each round takes the current 512-bit block being processed (Yq) and 128-bit buffer value ABCD and updates the contents of the buffer. Updating can be done as shown in Figure 2–18. Each round uses a 64-element static table T [1... 64] constructed from the sine function and 32-bit of a message block and a block of message denoted with X [i].

The output of the fourth round is added to the input to the first round to produce new input for next iteration. Addition operation is modulo of 2^{32}.

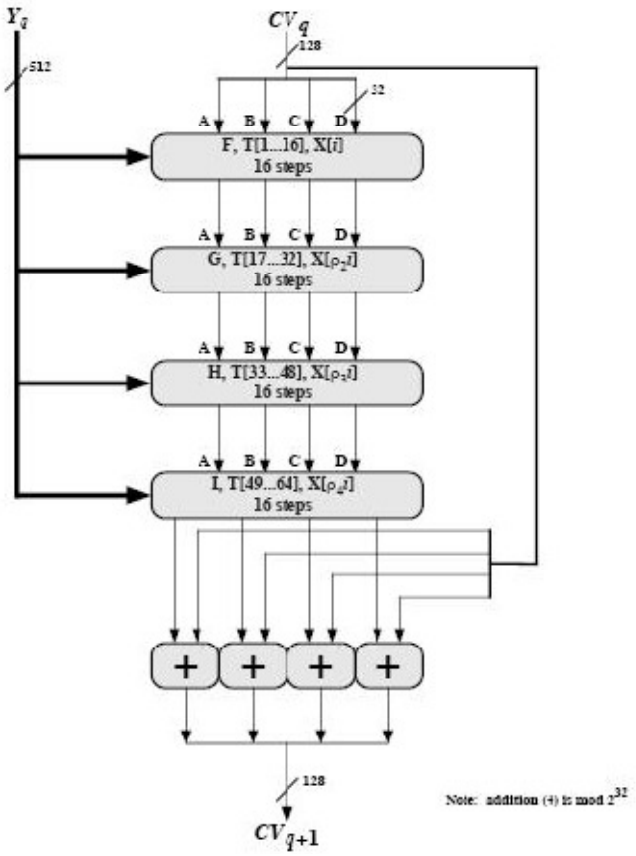


Figure 2.17 MD5 Processing of a single 512-bit block (Stallings, 2003, p.351)

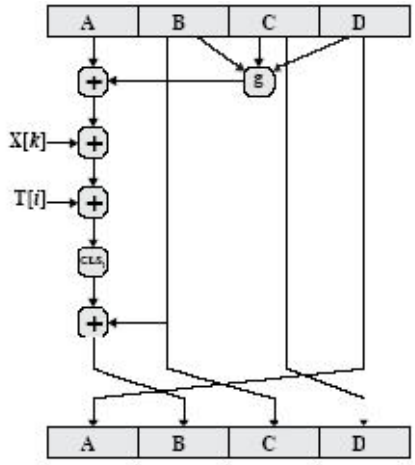


Figure 2.18. Updating of MD5, where g is one of the logical operations (F, G, H, I) (Stallings, 2003, p.353)

5. *Step 5: Output*. If all blocks of input are processed, the output of step 4 is equal to the output of the MD5 algorithm and it is the 128 bit message digest.

The MD5 algorithm is simple to implement, but it is not resistant to collisions. Oorschot and Wiener (Oorschot & Wiener, 1994) presented a design of a collision search machine for MD5 that could find a collision in 24 days. Dobbertin (Dobbertin, 1996) presents an attack which works on the operation of MD5 on a single 512-bit block and finds a collision easily.

Since the strength of hash functions against attacks depends on the length of produced hash code, a hash algorithm with longer hash code is needed. The longer code provides more secure system. Therefore, 128-bit hash code is not sufficient.

2.4.3.3 Secure Hash Algorithm (SHA)

SHA is a kind of message digest algorithm (or a hash function) designed by the NSA and published by the NIST. "A revised version was issued as a Federal Information Processing Standard (FIPS)" and it is known as SHA–1". It produces a 160-bit hash value from a message with a maximum size of 264 bits, and is based on some principles that are similar to MD5 algorithm. The input is processed in 512-bit blocks. The algorithm consists of the following steps and the details of which are given in.

1. *Step 1: Append padding bits.* The message is padded (expanded) to get the desired length which is an integer multiple of 512 bits. The number of padding bits can be in the range of 1 to 512. Padding starts with adding a single '1' bit and continues adding necessary number of '0' bits.

2. *Step 2: Append Length:* A block of 64 bits is appended to the message.

3. *Step 3: Initialize Message Digest buffer with hexadecimal numbers*. A 160-bit buffer contains five 32-bit registers called A, B, C, D, and E in which these values are stored in big-endian format, which is the most significant byte of 32-bit in the low-address byte position.

4. *Step 4: Process message in 512-bits blocks*. The SHA–1 function has four rounds of processing of 20 steps each. Rounds have a similar structure except using different logical functions. Each round takes the current 512-bit block being processed (Yq) and 160-bit buffer value ABCDE and updates the contents of the buffer. Each round uses a constant Kt where t is in the range of 0 to 80 and a 32- bit, Wt, which is derived from the current 512-bit input block as shown in Figure 2.19.

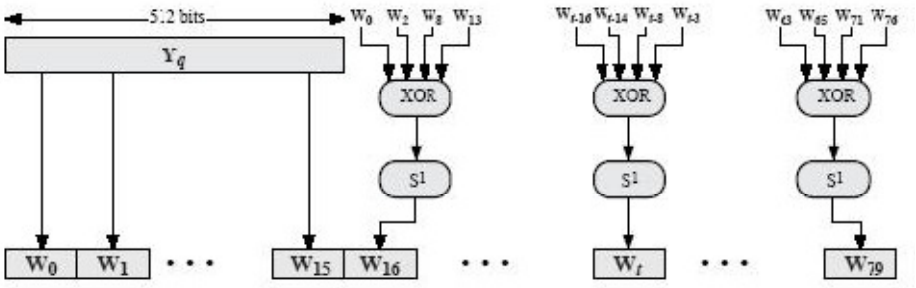


Figure 2.19 Creation of 80-word (Stallings, 2033, p.362)

This derivation technique makes difference between SHA -1 and MD5. "SHA–1 uses a linear recurrence to stretch the 16 words (a word is a 32-bit) of message block to the 80 words instead of processing each message block four times" (Ferguson & Schneir, 2003, p.88)

The output of the fifth round is added to the input to the first round to produce new input for next iteration. Addition operation is modulo of 232.

5. *Step 5: Output.* If all blocks of input are processed, the output of step 4 is equal to the output of the SHA–1 algorithm and it is the 160 bit message digest.

SHA–1 produce a hash value with 160-bit. It is obviously that it is more resistant to attacks than MD5 and slower than MD5. A difficulty of producing same hash value is on the order of 2160 operations, whereas after applying a birthday attack the number of operations decreases to 280. To overcome this sufficiency, NIST proposed variants of SHA–1 with -256, -384, and -512 bit digests. Three of them executes

more slowly than SHA–1 but more resistance to the attacks. Table 2.2 summarizes the properties of SHA- 1 and its variants.

Table 2.2 Comparison of SHA properties

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Maximum message	2^{64}	2^{64}	2^{128}	2^{128}
Block size	512	512	512	512
Number of steps	80	80	80	80
Security	80	128	192	256

2.4.3.4 RACE Integrity Primitives Evaluation Message Digest Algorithm 160 (RIPEMD–160)

RIPEMD is a kind of message digest algorithm (or hash function), designed by Hans Dobbertin, Antoon Bosselaers, and Bart Preneel (Bosselaers, Dobbertin, & Preneel, 1997). This group firstly developed a 128-bit version of RIPEMD; in 1994 the revised version was issued as RIPEMD–160 with a 160-bit hash value. It is intended to be used as a secure replacement for the MD5. It produces a 160-bit hash value from a message with any size. The input is processed in 512-bit blocks. The algorithm consists of the following steps and the details of which are given in (Stallings, 2003, p.365)

1. *Step 1: Append padding bits.* Padding bits are added in same way as done in both MD5 and SHA–1.

2. *Step 2: Append Length*: Length is added in same way as done in MD5.

3. *Step 3: Initialize Message Digest Buffer.* A 160-bit buffer, that contains five registers, is initialized with the same values as done in SHA–1.

4. *Step 4: Process message in 512-bits blocks.* The RIPEMD–160 function has ten

   rounds (two parallel lines of five rounds) of processing of 16 steps each as shown in Figure 2.20. Two parallel lines are used to increase the complexity of finding collisions. Rounds have a similar structure except using different logical functions. In the second five rounds, the functions are used in reverse order of first five rounds. Each round takes the current 512-bit block being processed (Yq) and 160-bit buffer value ABCDE and updates the contents of the buffer. Each round uses a constant Kj where j is in the range of 0 to 80 and a 32-bit of a message block. The output of the fifth round is added to the input to the first round to produce new input for next iteration. Addition operation is modulo of 2^{32}.

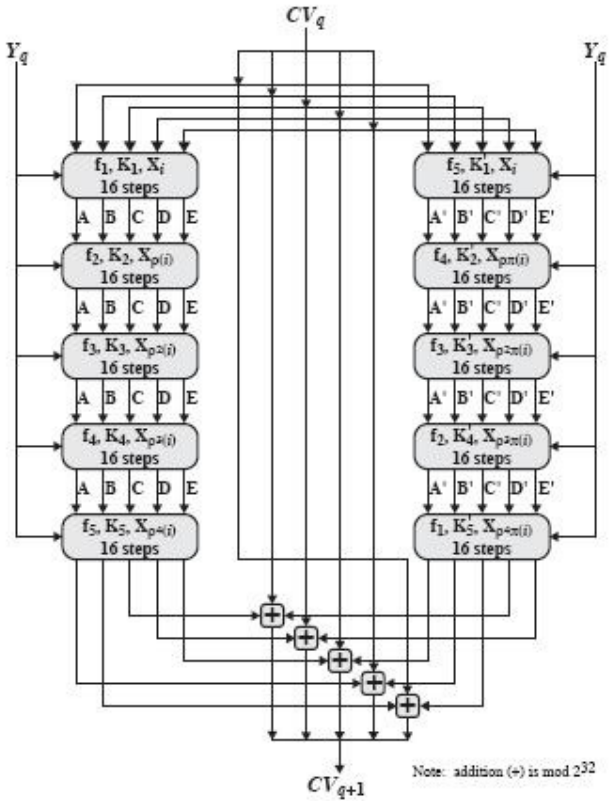


Figure 2.20 RIPEMD-160 Processing of a single 512-block (Stallings, 2003, p.366)

5. *Step 5: Output.* If all blocks of input are processed, the output of step 4 is equal to the output of the RIPEMD–160 algorithm and it is the 160 bit message digest. RIPEMD–256 and RIPEMD–320 are optional variants of RIPEMD–160, in which they have a longer hash result but do not provide a higher level of security than RIPEMD–160 (Dobbertin, Bosselaers & Preneel, 2004).

2.4.3.5 Comparison of Three Hash Functions

The mentioned hash functions are similar in many ways but have some differences that are given below:

1. *Resistant to attacks:* Both SHA–1 and RIPEMD–160 are more resistant to attacks than MD5 because of producing a longer hash value.

2. *Resistant to cryptanalysis:* Cryptanalysis of a hash function focuses on the internal structure of updating function. Both SHA–1 and RIPEMD–160 have a bit complex updating function than MD5. “The use of two lines of processing gives RIPEMD–160 added complexity, which should make cryptanalysis more difficult compared to SHA–1” (Stallings,2003,p.371)

3. *Speed and Performance:* Table 2.3 shows a set of results which compares the algorithms with respect to their speed.

Table 2.3 Performance analysis of several hash functions from (Dobbertin, Bosselaers & Preneel,, 2004)

Algorithm	Cycles	Megabit per second	Megabyte per second	Performance
MD5	337	136.7	17.09	0.72,
SHA–1	837	55.1	6.88	0.74,
RIPEMD–160	1033	45.5	5.68	0.24

CHAPTER THREE
OVERVIEW OF STEGANOGRAPHY

3.1 A Brief History of Steganography

The earliest recordings of Steganography were by the Greek historian Herodotus in his chronicles known as "Histories" and date back to around 440 BC. Herodotus recorded two stories of Steganographic techniques during this time in Greece. The first stated that King Darius of Susa shaved the head of one of his prisoners and wrote a secret message on his scalp. When the prisoner's hair grew back, he was sent to the Kings son in law Aristogoras in Miletus undetected. The second story also came from Herodotus, which claims that a soldier named Demeratus needed to send a message to Sparta that Xerxes intended to invade Greece. Back then, the writing medium was text written on wax-covered tablets. Demeratus removed the wax from the tablet, wrote the secret message on the underlying wood, recovered the tablet with wax to make it appear as a blank tablet and finally sent the document without being detected.

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

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

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

can be attributed to the foundation of some steganographic systems and more significantly to watermarking techniques.

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

In 1857, Brewster suggested hiding secret messages in spaces not larger than a full stop or small dot of ink. In 1860 the problem of making tiny images was solved by French photographer Dragon. During Franco – Prussian war (1870–1881) from besieged Paris messages were sent on microfilms using pigeon post. During Russo – Japanese war (1905) microscopic images were hidden in ears, nostrils, and under fingernails. During First World War messages to and from spies were reduced to microdots, by several stages of photographic reduction and then stuck on top of printed periods or commas (in innocuous cover materials, such as magazines).

Recently, the United States government claimed that Osama Bin Laden and the Al-Qaeda organization use Steganography to send messages through websites and newsgroups. However, until now, no substantial evidence supporting this claim has been found, so either al-Qaeda has used or created real good steganographic algorithms, or the claim is probably false.

Steganographic techniques have been used with success for centuries already. However, since secret information usually has a value to the ones who are not allowed to know it, there will be people or organizations who will try to decode encrypted information or find information that is hidden from them. Governments want to know what civilians or other governments are doing, companies want to be sure that trade secrets will not be sold to competitors and most persons are naturally curious. Many different motives exist to detect the use of Steganography, so techniques to do so continue to be developed while the hiding algorithms become

more advanced. With the research this topic is now getting we will see a lot of great applications for Steganography in the near future.

3.2 Different Kinds of Steganography

Almost all digital file formats can be used for Steganography, but the formats that are more suitable are those with a high degree of redundancy. Redundancy can be defined as the bits of an object that provide accuracy far greater than necessary for the object's use and display (Currie & Irvine, 1996). The redundant bits of an object are those bits that can be altered without the alteration being detected easily (Anderson & Petitcolas, 1998). Image and audio files especially comply with this requirement, while research has also uncovered other file formats that can be used for information hiding.

Figure 1 shows the four main categories of file formats that can be used for steganography.

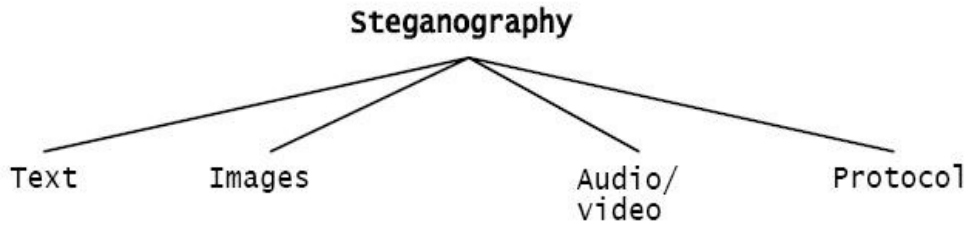


Figure 3.1 Categories of steganography

Data may be embedded in files at imperceptible levels as noise. Properties of images can be manipulated including luminescence, contrast and colors (Johnson, Zoran & Sushil, 2001). In audio files small echoes or slight delays can be included or subtle signals can be masked with sounds of higher amplitude. Information can be hidden in documents by manipulating the positions of the lines or the words. When HTML files are written web browsers ignore spaces, tabs, certain characters and extra line breaks. These could be used as locations in which to hide information. Messages can be retrieved from text by taking for example the second letter of each

word and using them to produce the hidden message. This is called a null cipher or open code (Johnson, Zoran & Sushil, 2001). Information can be hidden in the layout of a document for example certain words in a piece of text can be shifted very slightly from their positions and these shifted words can then make up the hidden message. The way a language is spoken can be used to encode a message such as pauses, enunciations and throat clearing (Johnson, Zoran & Sushil, 2001).

Unused or reserved space on a disc can be used to hide information. The way operating systems store files typically results in unused space that appears to be allocated to the files. A minimum amount of space may be allocated to files but the file does not need all this space so some of it goes unused. This space can be used to hide information. Another method for hiding information in file systems is to create a hidden partition (Johnson, Zoran & Sushil, 2001). Data may be hidden in unused space in file headers. Packets for example TCP / IP packets have headers with unused space and other features that can be manipulated to embed information (Johnson, Zoran & Sushil, 2001). Data can be hidden using the physical arrangement of a carrier for example the layout of code in a program or electronic circuits on a board. This process can be used to record and identify the origin of the design and cannot be removed without a substantial change to the physical layout. Spread spectrum techniques can also be used by placing an audio signal over a number of different frequencies. Random number generators are developed to allow spread spectrum radios to hop from frequency to frequency. Systems can use different frequencies at the same time. Some information is broadcast on one frequency and some on another. The message can be reassembled by combining all the information (Wayner, 2002).

3.2.1 Encoding Secret Messages in Text

Encoding secret messages in text can be a very challenging task. This is because text files have a very small amount of redundant data to replace with a secret message. Another drawback is the ease of which text based Steganography can be altered by unwanted parties by just changing the text itself or reformatting the text to

some other form (from. TXT to. PDF, etc.). There are numerous methods by which to accomplish text based Steganography. I will introduce a few of the more popular encoding methods below.

Line-shift encoding involves actually shifting each line of text vertically up or down by as little as 3 centimeters. Depending on whether the line was up or down from the stationary line would equate to a value that would or could be encoded into a secret message.

Word-shift encoding works in much the same way that line-shift encoding works, only we use the horizontal spaces between words to equate a value for the hidden message. This method of encoding is less visible than line-shift encoding but requires that the text format support variable spacing.

Feature specific encoding involves encoding secret messages into formatted text by changing certain text attributes such as vertical/horizontal length of letters such as b, d, T, etc. This is by far the hardest text encoding method to intercept as each type of formatted text has a large amount of features that can be used for encoding the secret message.

All three of these text based encoding methods require either the original file or the knowledge of the original files formatting to be able to decode the secret message.

3.2.2. Encoding Secret Messages in Audio

Encoding secret messages in audio is the most challenging technique to use when dealing with Steganography. This is because the human auditory system (HAS) has such a dynamic range that it can listen over. To put this in perspective, the (HAS) perceives over a range of power greater than one million to one and a range of frequencies greater than one thousand to one making it extremely hard to add or remove data from the original data structure. The only weakness in the (HAS) comes

at trying to differentiate sounds (loud sounds drown out quiet sounds) and this is what must be exploited to encode secret messages in audio without being detected.

There are two concepts to consider before choosing an encoding technique for audio. They are the digital format of the audio and the transmission medium of the audio.

There are three main digital audio formats typically in use. They are Sample Quantization, Temporal Sampling Rate and Perceptual Sampling.

Sample Quantization which is a 16-bit linear sampling architecture used by popular audio formats such as (.WAV and. AIFF). Temporal Sampling Rate uses selectable frequencies (in the KHz) to sample the audio. Generally, the higher the sampling rate is, the higher the usable data space gets. The last audio format is Perceptual Sampling. This format changes the statistics of the audio drastically by encoding only the parts the listener perceives, thus maintaining the sound but changing the signal. This format is used by the most popular digital audio on the Internet today in ISO MPEG (MP3).

Transmission medium (path the audio tapes from sender to receiver) must also be considered when encoding secret messages in audio. W. Bender introduces four possible transmission mediums:

1) Digital end to end - from machine to machine without modification.
2) Increased/decreased resampling - the sample rate is modified but remains digital.
3) Analog and resampled - signal is changed to analog and resampled at a different rate.
4) Over the air - signal is transmitted into radio frequencies and resampled from a microphone.

We will now look at three of the more popular encoding methods for hiding data inside of audio. They are low-bit encoding, phase-coding and spread spectrum.

Low-bit encoding embeds secret data into the least significant bit (LSB) of the audio file. The channel capacity is 1KB per second per kilohertz (44 kbps for a 44 KHz sampled sequence). This method is easy to incorporate but is very susceptible to data loss due to channel noise and resampling.

Phase coding substitutes the phase of an initial audio segment with a reference phase that represents the hidden data. This can be thought of, as sort of an encryption for the audio signal by using what is known as Discrete Fourier Transform (DFT), which is nothing more than a transformation algorithm for the audio signal.

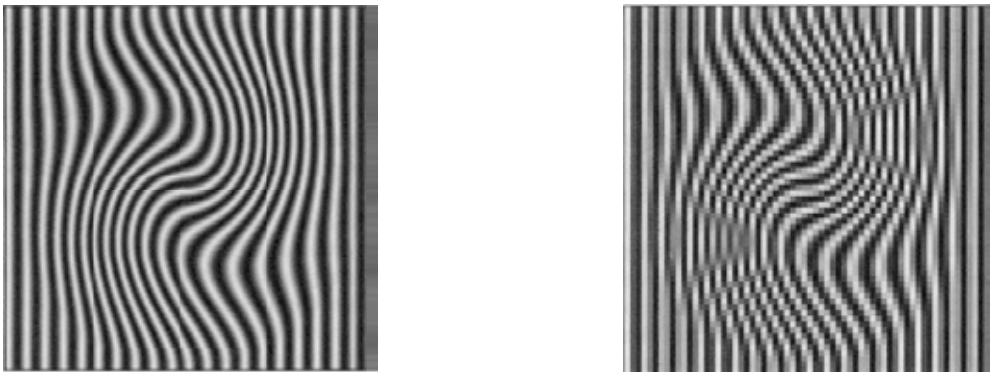
Spread spectrum encodes the audio over almost the entire frequency spectrum. It then transmits the audio over different frequencies which will vary depending on what spread spectrum method is used. Direct Sequence Spread Spectrum (DSSS) is one such method that spreads the signal by multiplying the source signal by some pseudo random sequence known as a (CHIP). The sampling rate is then used as the chip rate for the audio signal communication. Spread spectrum encoding techniques are the most secure means by which to send hidden messages in audio, but it can introduce random noise to the audio thus creating the chance of data loss.

3.2.3 Encoding Secret Messages in Image

3.2.3.1 Image Structure and Image Processing

A digital image is the most common type of carrier used for steganography. A digital image is produced using a camera, scanner or other device. The digital representation is an approximation of the original image (Efford & Nick, 2000). The system used for producing the image focuses a two dimensional pattern of varying light intensity and color onto a sensor (Efford & Nick, 2000). The pattern has a co-ordinate system and the origin is the upper left hand corner of the image. The pattern can be described by a function f(x, y). An image can be described as an array of numbers that represent light intensities at various points. These light intensities or

instances of color are called pixels. Sampling is the process of measuring the value of the image function f(x, y) at discrete intervals in space (Efford & Nick, 2000). Each sample is the small square area of the image known as the pixel. The raster data of an image is that part of the image that can be seen i.e. the pixels (Johnson, Zoran & Sushil, 2001). The size of an image can be given in pixels, for example an image which is 640 x 480 pixels contains 307,200 pixels (Johnson, Zoran & Sushil, 2001) Pixels are indexed by x and y co-ordinates with x and y having integer values (Efford & Nick, 2000). The spatial resolution of an image is the physical size of the pixel in the image. Dense sampling produces a high-resolution image in which there are many pixels and each contributes a small part of the scene. Coarse sampling results in a low-resolution image in which there are fewer pixels (Efford & Nick, 2000). The rate of change of the value f(x, y) as it moves across the image is the spatial frequency. Gradual changes in f(x, y) correspond to low spatial frequencies and can be represented in a coarsely sampled image. Rapid changes correspond to high spatial frequencies and must be represented by a densely sampled image. The Nyquist criterion states that the sampling frequency should be at least double the highest spatial frequency found in the image. A coarsely sampled image that does not follow this criterion may suffer from the effects of aliasing (Efford & Nick, 2000) shown in Figure 3.2 below.



a. b.

Figure 3.2 a. Image without aliasing b. coarsely sampled image showing aliasing artefacts (Efford & Nick, 2000).

Each pixel is generally stored as 24-bit or 8-bit. A 24-bit pixel has a possibility of 224 color combinations (Johnson, Zoran & Sushil, 2001) The 24 bits of a 24-bit image are spread over three bytes and each byte represents red, green and blue respectively. Colors are obtained by mixing red, green and blue light in different proportions. An image can be formed by making three measurements of brightness at each pixel using the red, green and blue components of the detected light. Using the RGB model the value of f(x, y) is a vector with three components corresponding to red (R), green (G) and blue (B). They can be regarded as orthogonal axes defining a three dimensional color space. Every value of f(x, y) is a point in the color cube shown in Figure 3.3 below. The three components are normally quantized using 8 bits. An image made of these components is described as a 24-bit color image (Efford & Nick, 2000).

Each byte can have a value from 0 to 255 representing the intensity of the color. The darkest color value is 0 and the brightest is 255. For example a pixel could be made up of three bytes as follows: 11111111 00000000 00000000. The first 8 bits represent red, the second 8 bits represent green and the third 8 bits represent blue. The bit values in this example result in a red pixel. Its red byte is at a maximum value (11111111) and its green (00000000) and blue (00000000) bytes have the lowest possible value. Transparency is controlled by the addition of information to each element of the pixel data. This is to allow image overlay (Murray & William, 1996). A 24-bit pixel value can be stored in 32 bits. The extra 8 bits specify transparency. This is sometimes called the alpha channel (Murray & William, 1996). An ideal 8-bit alpha channel can support transparency levels from 0 (completely transparent) to 255 (completely opaque). It can be stored as part of the pixel data (Murray & William, 1996) e.g. RGBA (red, green, blue and alpha taking up 4 bytes in total).

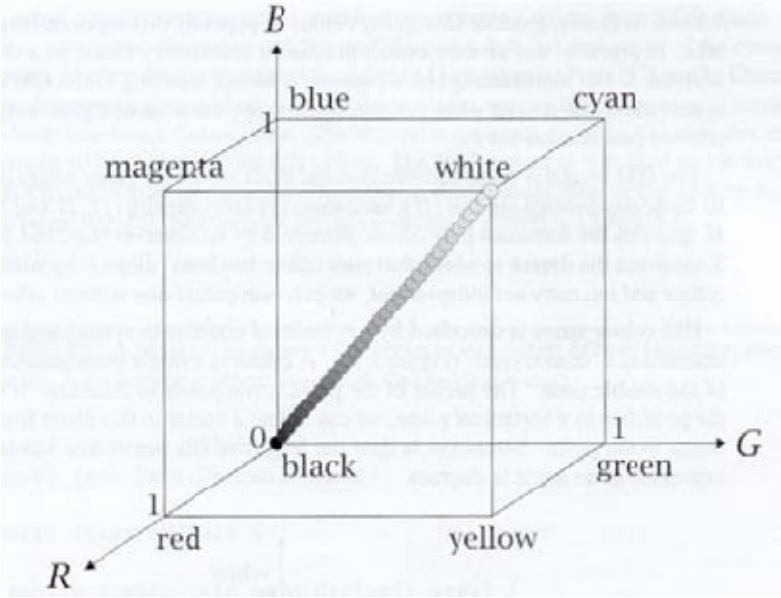


Figure 3.3 The RGB color cube (Efford & Nick, 2000)

Some images are 8-bit. Each pixel is represented by one byte only. This one byte can have any value ranging from 0 to 255, 256 possible colors or 256 grayscale values for black and white images. The colors are taken from a color index or palette also called a color map or color table. This palette contains up to 256 colors representing the colors in the image. The value of the pixel in an image points to a color in the palette (Johnson, Zoran & Sushil, 2001). The GIF (Graphic Interchange Format) image format uses this process. When a GIF image is displayed the software paints the specified color from the palette onto the screen (Johnson, Zoran & Sushil, 2001) at each pixel. If the image has fewer colors than the size of the palette any unused colors in the palette are set to zero (Murray & William, 1996) GIF is a bitmap image. Bitmap is a system in which an image is described as a bit pattern or series of numbers that gives the shade or color of each pixel (Day & Knudsen, 1998). In true grayscale images, values from 0 to 255 represent the intensity of the color and do not refer to a palette (Johnson, Zoran & Sushil, 2001). A palette image format contains a header, a palette and image data (pointers to the palette).

There are two steps in creating palette-based images - color quantization and dithering (Fridrich, 1999). A color quantization algorithm has two parts, generating the color palette and mapping the pixels. To generate the palette, colors are extracted

from the image. Each pixel in the image is then mapped to its nearest color in the palette to generate the quantized image. Quantization involves replacing a continuously varying f(x, y) with a set of quantization levels (Efford & Nick, 2000). A set of quantization levels comprises the integers 0, 1, 2, 3… n-1. 0 and n-1 are displayed as black and white with intermediate levels in shades of grey (grayscale). The number of grey levels is usually an integral power of 2. n = 2b where b is the number of bits used for quantization. It is typically 8 resulting in images with 256 grey levels ranging from black to white (Efford & Nick, 2000). There are different algorithms for quantization. Color quantization involves truncating all the colors of the original 24-bit image to a finite number of colors, 256 for GIF, 216 for Netscape GIF and 2 for black and white. Splitting algorithms split the color space of the original image into two subspaces according to some preference criteria. The splitting is iteratively carried out until the correct number of subspaces is reached. The color representing the subspace becomes the quantized color. Clustering methods can also be used in which colors are clustered to form the quantized colors. The method usually used for quantization involves iterative dividing of a three dimensional color cube into two boxes with approximately the same number of colors. The half with the largest dimensions is chosen by measuring either the greatest difference in RGB value or the greatest difference in luminosity (Wayner, 2002). The half with the largest dimensions is selected and the iteration is continued until the desired number of colors is produced (Fridrich, 1999). The centers of gravity of each box are then rounded to integer colors representing the colors of the palette. The largest dimension can be replaced using the largest standard deviation resulting in a slightly better algorithm (Fridrich, 1999). It should be noted that standard quantization algorithms will not necessarily yield exactly 256 colors in the image.

Dithering is a technique used to simulate colors that are missing from an images palette. This is done by intermingling pixels of two or more palette colors. Colors are reordered so that their visual combination matches the original images more closely. If the unavailable color differs too much from the colors in the palette a grainy appearance results (Johnson, Zoran & Sushil, 2001) and errors are present in the

form of false contours. Dithering can be based on error diffusion. The image is scanned in some regular way for example by rows. The color of a pixel is rounded to its closest color in the palette. This produces an error. The error is negative if the rounding has resulted in a decrease in the pixel value. The error is positive if the rounding has increased the pixel value. The error is multiplied by weights and added to the surrounding pixels that have not yet been visited. Using this method the rounding error is spread to neighboring pixels which results in an image that is more visually pleasing (Fridrich, 1999). This process is repeated for every color in the image (Johnson, Zoran & Sushil, 2001).

3.2.3.2 Image Compression

When working with larger images of greater bit depth, the images tend to become too large to transmit over a standard Internet connection. In order to display an image in a reasonable amount of time, techniques must be incorporated to reduce the image's file size. These techniques make use of mathematical formulas to analyze and condense image data, resulting in smaller file sizes. This process is called compression (Anderson & Petitcolas, 1998).

In images there are two types of compression: lossy and lossless. Both methods save storage space, but the procedures that they implement differ. Lossy compression creates smaller files by discarding excess image data from the original image. It removes details that are too small for the human eye to differentiate, resulting in close approximations of the original image, although not an exact duplicate. An example of an image format that uses this compression technique is JPEG (Joint Photographic Experts Group) (Johnson & Jajodia, 1998).

Lossless compression, on the other hand, never removes any information from the original image, but instead represents data in mathematical formulas. The original image's integrity is maintained and the decompressed image output is bit-by-bit

identical to the original image input. The most popular image formats that use lossless compression is GIF (Graphical Interchange Format) and 8-bit BMP (a Microsoft Windows bitmap file) (Johnson & Jajodia, 1998).

Compression plays a very important role in choosing which steganographic algorithm to use. Lossy compression techniques result in smaller image file sizes, but it increases the possibility that the embedded message may be partly lost due to the fact that excess image data will be removed (Dunbar, 2002). Lossless compression though, keeps the original digital image intact without the chance of lost, although it does not compress the image to such a small file size (Johnson & Jajodia, 1998). Different steganographic algorithms have been developed for both of these compression types and will be explained in the following sections.

3.2.3.3 Image and Transform Domain

Image steganography techniques can be divided into two groups: those in the Image Domain and those in the Transform Domain (Silman, 2001). Image – also known as spatial – domain techniques embed messages in the intensity of the pixels directly, while for transform – also known as frequency – domain, images are first transformed and then the message is embedded in the image (Lee & Chen, 2000).

Image domain techniques encompass bit-wise methods that apply bit insertion and noise manipulation and are sometimes characterized as “simple systems” (Johnson & Jajodia, 1998). The image formats that are most suitable for image domain steganography are lossless and the techniques are typically dependent on the image format (Venkatraman, Abraham & Paprzycki, 2004).

Steganography in the transform domain involves the manipulation of algorithms and image transforms (Johnson & Jajodia, 1998). These methods hide messages in more significant areas of the cover image, making it more robust (Wang, H. & Wang, S., 2004). Many transform domain methods are independent of the image

format and the embedded message may survive conversion between lossy and lossless compression (Venkatraman, Abraham & Paprzycki, 2004).

In the next sections steganographic algorithms will be explained in categories according to image file formats and the domain in which they are performed.

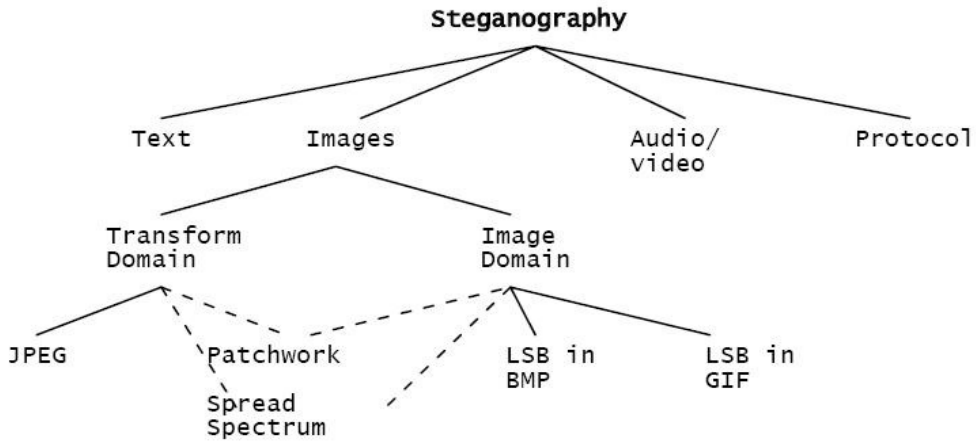


Figure 3.4.Techniques use for Steganography.

3.2.3.3.1 JPEG Steganography. Originally it was thought that steganography would not be possible to use with JPEG images, since they use lossy compression which results in parts of the image data being altered. One of the major characteristics of steganography is the fact that information is hidden in the redundant bits of an object and since redundant bits are left out when using JPEG it was feared that the hidden message would be destroyed. Even if one could somehow keep the message intact it would be difficult to embed the message without the changes being noticeable because of the harsh compression applied. However, properties of the compression algorithm have been exploited in order to develop a steganographic algorithm for JPEGs.

One of these properties of JPEG is exploited to make the changes to the image invisible to the human eye. During the DCT transformation phase of the compression algorithm, rounding errors occur in the coefficient data that are not noticeable (Johnson & Jajodia, 1998). Although this property is what classifies the algorithm as being lossy, this property can also be used to hide messages.

It is neither feasible nor possible to embed information in an image that uses lossy compression, since the compression would destroy all information in the process. Thus it is important to recognize that the JPEG compression algorithm is actually divided into lossy and lossless stages. The DCT and the quantization phase form part of the lossy stage, while the Huffman encoding used to further compress the data is lossless. Steganography can take place between these two stages. Using the same principles of LSB insertion the message can be embedded into the least significant bits of the coefficients before applying the Huffman encoding. By embedding the information at this stage, in the transform domain, it is extremely difficult to detect, since it is not in the visual domain.

3.2.3.3.2 Patchwork. Patchwork is a statistical technique that uses redundant pattern encoding to embed a message in an image (Johnson & Jajodia, 1998). The algorithm adds redundancy to the hidden information and then scatters it throughout the image (Johnson & Jajodia, 1998). A pseudorandom generator is used to select two areas of the image (or patches), patch A and patch B (Bender, Gruhl, Morimoto, & Lu, 1996). All the pixels in patch A is lightened while the pixels in patch B is darkened (Bender, & others, 1996). In other words the intensities of the pixels in the one patch are increased by a constant value, while the pixels of the other patch are decreased with the same constant value (Marvel, Boncelet, & Retter, 1999) The contrast changes in this patch subset encodes one bit and the changes are typically small and imperceptible, while not changing the average luminosity (Johnson & Jajodia, 1998).

A disadvantage of the patchwork approach is that only one bit is embedded. One can embed more bits by first dividing the image into sub-images and applying the embedding to each of them (Petitcolas, Anderson, & Kuhn, 1999). The advantage of using this technique is that the secret message is distributed over the entire image, so should one patch be destroyed, the others may still survive (Johnson & Jajodia, 1998). This however, depends on the message size, since the message can only be

repeated throughout the image if it is small enough. If the message is too big, it can only be embedded once (Johnson & Jajodia, 1998).

The patchwork approach is used independent of the host image and proves to be quite robust as the hidden message can survive conversion between lossy and lossless compression (Petitcolas, Anderson, & Kuhn, 1999).

3.2.3.3.3 Spread Spectrum. In spread spectrum techniques, hidden data is spread throughout the cover-image making it harder to detect (Wang, H., & Wang, S., 2004). A system proposed by Marvel et al. combines spread spectrum communication, error control coding and image processing to hide information in images (Marvel, Boncelet, & Retter, 1999)

Spread spectrum communication can be defined as the process of spreading the bandwidth of a narrowband signal across a wide band of frequencies (Marvel, Boncelet, & Retter, 1999). This can be accomplished by adjusting the narrowband waveform with a wideband waveform, such as white noise. After spreading, the energy of the narrowband signal in any one frequency band is low and therefore difficult to detect (Marvel, Boncelet, & Retter, 1999). In spread spectrum image steganography the message is embedded in noise and then combined with the cover image to produce the stego image. Since the power of the embedded signal is much lower than the power of the cover image, the embedded image is not perceptible to the human eye or by computer analysis without access to the original image (Marvel, Boncelet, & Retter, 1999).

3.2.3.3.4 LSB and Palette Based Images. Palette based images, for example GIF images, are another popular image file format commonly used on the Internet. By definition a GIF image cannot have a bit depth greater than 8, thus the maximum number of colors that a GIF can store is 256. GIF images are indexed images where the colors used in the image are stored in a palette, sometimes referred to as a color lookup table. Each pixel is represented as a single byte and the pixel data is an index to the color palette (Johnson & Jajodia, 1998). The colors of the palette are typically

ordered from the most used color to the least used colors to reduce lookup time (Johnson & Jajodia, 1998).

GIF images can also be used for LSB steganography, although extra care should be taken. The problem with the palette approach used with GIF images is that should one change the least significant bit of a pixel, it can result in a completely different color since the index to the color palette is changed (Johnson & Jajodia, 1998). If adjacent palette entries are similar, there might be little or no noticeable change, but should the adjacent palette entries be very dissimilar, the change would be evident (Johnson & Jajodia, 1998). One possible solution is to sort the palette so that the color differences between consecutive colors are minimized (Chandramouli, Kharrazi, & Memon, 2003). Another solution is to add new colors which are visually similar to the existing colors in the palette. This requires the original image to have less unique colors than the maximum number of colors (this value depends on the bit depth used) (Moerland, n.d.). Using this approach, one should thus carefully choose the right cover image. Unfortunately any tampering with the palette of an indexed image leaves a very clear signature, making it easier to detect.

A final solution to the problem is to use grayscale images. In an 8-bit grayscale GIF image, there are 256 different shades of grey (Johnson & Jajodia, 1998). The changes between the colors are very gradual, making it harder to detect.

3.2.3.4.5 Masking and Filtering. Masking and filtering techniques are mostly used on 24 bit and grayscale images. They hide info in a way similar to watermarks on actual paper and are sometimes used as digital watermarks. Masking images entails changing the luminance of the masked area. The smaller the luminance change, the less of a chance that it can be detected.

Stego-images (images that have been manipulated by steganographic methods) that are masked will keep a higher fidelity than LSB through compression, cropping and some image processing. The reason that a stego image encoded with masking, degrades less under JPEG compression is that the secret message is hid in the significant areas of the picture. There is a tool called JPEG – J steg that takes

advantage of the compression of JPEG while trying to keep high message fidelity. The program takes a secret message and a lossless cover image as input and outputs a stego image in JPEG format.

3.2.3.4.6 Digital Watermarking. As stated above digital watermarking is often performed by masking. The reason for digital watermarking is very different from steganography. Where the goal of steganography is to transmit a message undetected, a digital watermark is created as a sign of ownership/authorship. Since digital copies are inherently exact replicas of the original unless noise or some type of lossy operation is performed, there will be no way to tell them apart. Therein lays the authorship/ownership problem due to the likeness of the original and the copy. Digital watermarks can be used to show proof of ownership by having your mark put into the file, so even if both images are the same, if they contain your mark then you have a much stronger case for copyright or ownership disputes. Watermarks can be visible or invisible depending on the luminance in the mask. The higher the luminance the greater the visibility of the watermark. Attackers can use different types of image processing to remove or degrade the watermark until it is illegible. There are different recovery techniques but it is usually helpful to have the original image when trying to recover the watermark.

3.2.3.4.7 Redundant Pattern Encoding. Patchwork and other similar tools do redundant pattern encoding, which is a sort of spread spectrum technique. It works by scattering the message throughout the picture. This makes the image more resistant to cropping and rotation. Smaller secret images work better to increase the redundancy embedded in the cover image, and thus make it easier to recover if the stego-image is manipulated.

3.2.3.4.8 Encrypt and Scatter. The encrypt and scatter technique tries to emulate white noise. White Noise Storm is one such program that employs spread spectrum and frequency hopping. It does this by scattering the message throughout an image on eight channels within a random number that is generated by the previous window size and data channel. The channels then swap rotate, and interlace amongst each other. Each channel represents one bit and as a result there are many unaffected bits

in each channel. This technique is a lot harder to extract a message out of than an LSB scheme because to decode you must first detect that a hidden image exists and extract the bit pattern from the file. While that is true for any stegoimage you will also need the algorithm and stego key to decode the bit pattern, both of which are not required to recover a message from LSB. Some people prefer this method due to the considerable amount of extra effort that someone without the algorithm and stego-key would have to go through to extract the message. Even though White Noise Storm provides extra security against message extraction it is just as susceptible as straight LSB to image degradation due to image processing.

3.2.3.4.9 Least Significant Bit (LSB). Least significant bit (LSB) insertion is a common approach to embedding information in a cover image (Johnson, Zoran & Sushil, 2001). The least significant bit (in other words, the 8th bit) of some or all of the bytes inside an image is changed to a bit of the secret message. When using a 24-bit image, a bit of each of the red, green and blue color components can be used, since they are each represented by a byte. In other words, one can store 3 bits in each pixel. An 800 × 600 pixel image, can thus store a total amount of 1,440,000 bits or 180,000 bytes of embedded data (Krenn, 2004.). For example a grid for 3 pixels of a 24-bit image can be as follows:

(00101101 00011100 11011100)
(10100110 11000100 00001100)
(11010010 10101101 01100011)

When the number 200, which binary representation is 11001000, is embedded into the least significant bits of this part of the image, the resulting grid is as follows:

(00101101 00011101 11011100)
(10100110 11000101 00001100)
(11010010 10101100 01100011)

Although the number was embedded into the first 8 bytes of the grid, only the 3 underlined bits needed to be changed according to the embedded message. On average, only half of the bits in an image will need to be modified to hide a secret

message using the maximum cover size (Krenn, 2004). Since there are 256 possible intensities of each primary color, changing the LSB of a pixel results in small changes in the intensity of the colors. These changes cannot be perceived by the human eye - thus the message is successfully hidden. With a well-chosen image, one can even hide the message in the least as well as second to least significant bit and still not see the difference (Johnson, Zoran & Sushil, 2001).

In the above example, consecutive bytes of the image data – from the first byte to the end of the message – are used to embed the information. This approach is very easy to detect (Wang, H. & Wang, S., 2004). A slightly more secure system is for the sender and receiver to share a secret key that specifies only certain pixels to be changed.

The simple model of a stegosystem is shown as below.

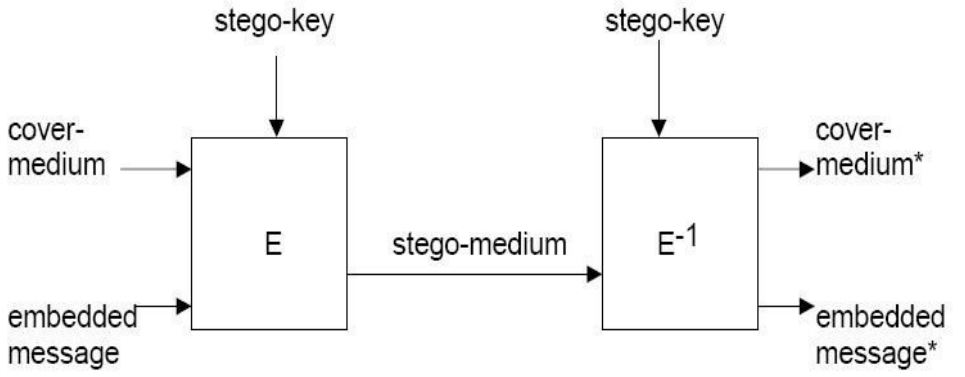


Figure 3.6 Model of a stegosystem.

Here is the model of a cryptosystem and model of a stegosystem strengthened with crypto operations.

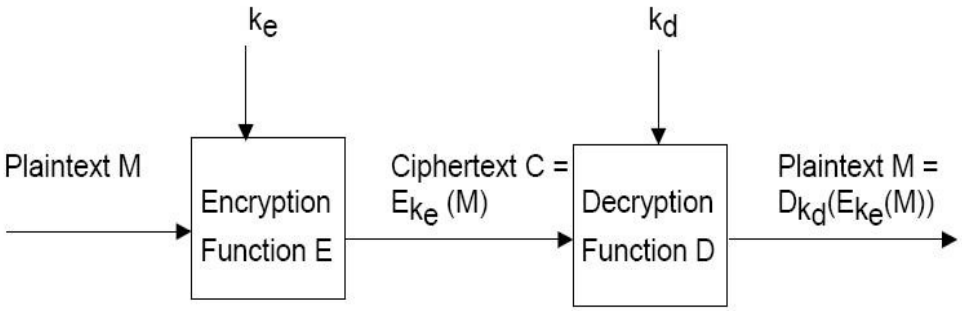


Figure 3.7 Model of a cryptosystem.

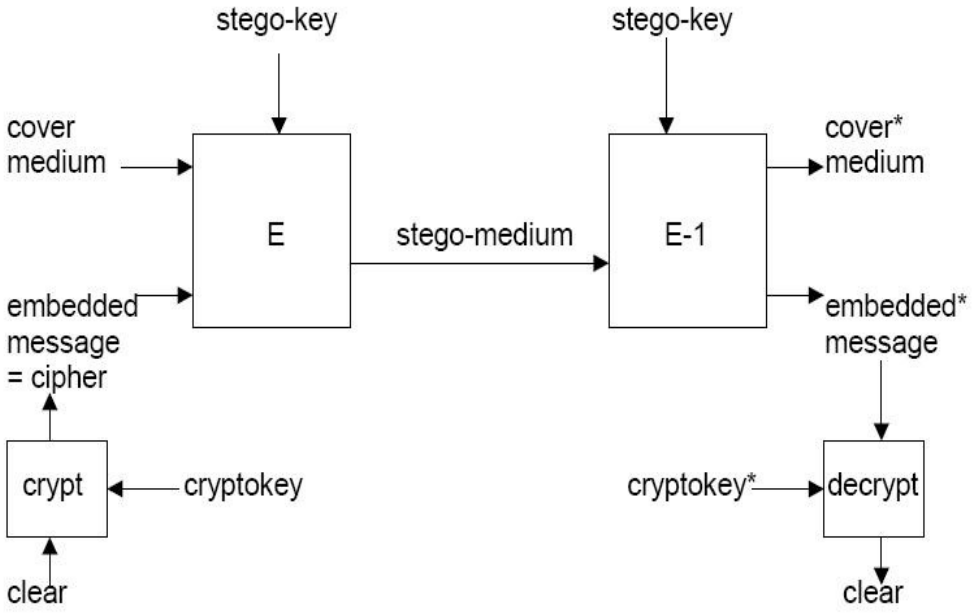


Figure 3.8 Combination of cryptography and steganography

CHAPTER FOUR
STEGANOGRAPHY DETECTING TECHNIQUES

4.1 Introduction

Replacement of redundant bits can change the statistical properties of the cover medium. Hence, attacks may detect the existence of hidden message. These attacks are also known as 'Steganalysis'. Steganalysis is defined as the art and science of breaking the security of steganographic systems. "Two aspects of attacks on steganography are detection and destruction of the embedded message. Any image can be manipulated with the intent of destroying some hidden information whether an embedded message exists or not" (Johnson & Jajodia, 1998). In a stego-system many attacks can be defined as shown in Figure 4.1.

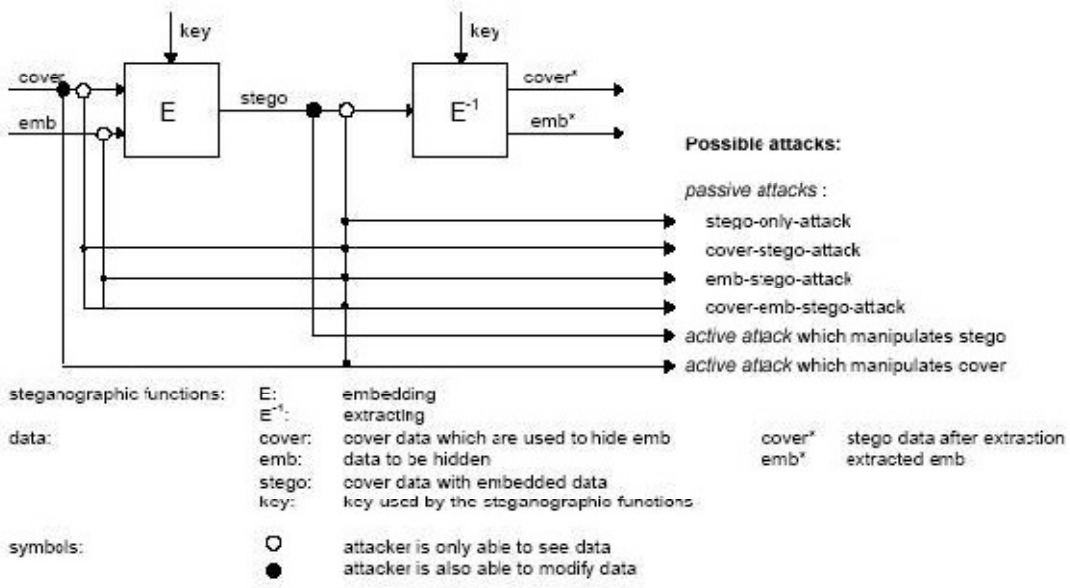


Figure 4.1 Possible attacks on a stego system (Franz & Pfitzmann, 2000)

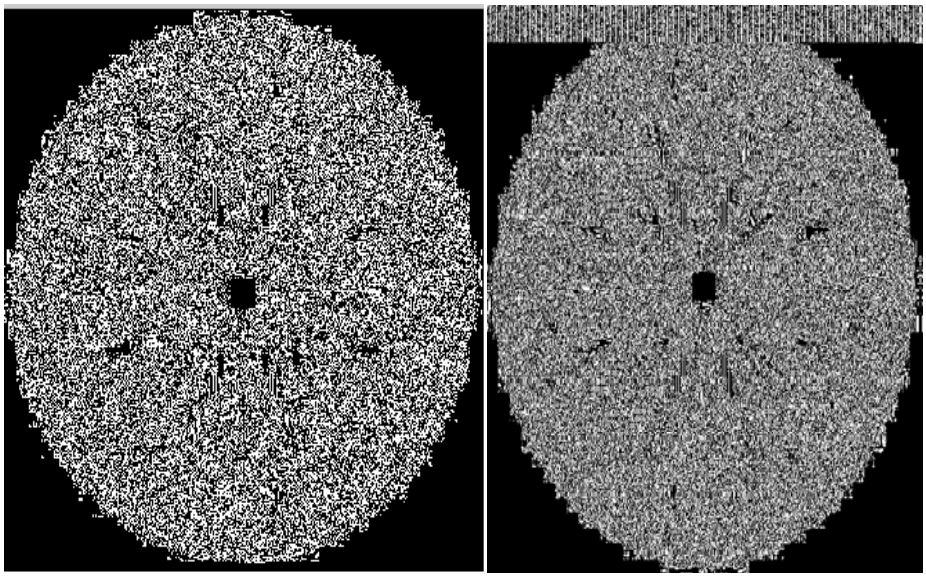
A stego-only-attack occurs when only the stego-image is available to be analyzed. If a cover image is also available to the attacker, this attack is known as cover-stego attack (also called known cover attack). If a hidden message is revealed at some later date the attacker could analyze the stego-image for future attacks. This is called

emb-stegoattack (also called known message attack). In cover-emb-stego-attack, the attacker knows cover and stego image and emb. These all are called passive attacks. In passive attacks, attacker can only be able to see data. Furthermore, the attacker can be able to manipulate cover or stego. In the first one, the steganalyst generates stego-image from some steganography tool from a known message. In the second one, stego-image can be manipulated to prevent the transmission of the embedded message.

The-Stego-Only-Attack is the most important and common attack. In this attack two different approaches can be distinguished: HVS ability is used for the visual attacks and the statistical tests on the stego file is used for statistical attacks.

4.2 Visual Attacks

Many steganographic tools assume that “least significant bits of luminance values in digital images are completely random and could therefore be replaced” (Westfeld & Pfitzmann, 1999) by message bits. But, visual attacks reveal that this assumption is wrong. Sometimes the LSBs are not very random. With a naked eye, a human may distinguish whether there is a message distorting the image content or not. The filtering process which can be used for this purpose involves removing all parts of the image covering message. Filtered image reveals whether any message is hidden or not. Figure 4.2 a shows the filtered cover image and there is nothing unusual, but Figure 4.2 b shows filtered stego image and it is obvious that approximately 10% of image contains a hidden message. A human eye can easily recognize the image that contains data.



(a) (b)

Figure 4.2 (a) Nothing Embedded (b) approximately 10% capacity is embedded

The visual attack visually represents the steganographic values of the pixels in an image. If the steganographic system simply overwrites the LSB of the pixel's sequence in a grey scale image (8-bit), the filtered one displays a black pixel whenever a pixel value is even and a white when it is odd. This means that the potential message is removed from the image. If it looks like some remaining image content, it can be said that nothing is embedded.

Same procedure is applied for a 24-bit image by repeating the LSB in the whole byte. If a 24 bit sample looks like this: “???????a???????b???????c”, the filtered state looks like this: ”aaaaaaaabbbbbbbbcccccccc”.

4.3 Statistical Attacks

“Visual attacks have two drawbacks” (Hetzl, 2002). Every image must be filtered, displayed and evaluated by a human eye. These processes are very time consuming. Besides that, some original images might contain random data in LSBs and a visual attack behaves this cover image as a stego image. One additional drawback occurs when JPEG images are used as a cover image. Visual

attacks can easily detect the images whose LSBs are modified. “This is not true for the JPEG format. The modifications happen in the frequency domain instead of the spatial domain, so there are no visual attacks against the JPEG image format” (Provos & Honeyman, 2002).

The premise of statistical attacks is changes in histogram of color frequencies. “Pfitzmann and Westfeld introduce a method based on statistical analysis of pair of values (PoV) exchanged during message embedding” (Fridrich et al, 2001). PoV means two adjacent color values. “If the bits used for overwriting the least significant bits are equally distributed, the frequencies of both values of each PoV become equal.” (Wesfeld & Pfitzmann, 1999). This statistical attack performs the following steps:

1. Frequency histogram of color values (or DCT coefficients for a JPEG image) is, calculated

2. Theoretically expected frequency in category ‘i’ after embedding is calculated by taking arithmetic mean. Let it be the color (or DCT) histogram.

$$y_i^* = \frac{n_{2i}+n_{2i+1}}{2}$$ (Provos, 2001)

3. The observed frequency is calculated as follows:

$$y_i = n_{2i}$$ (Provos, 2001)

4. The degree of similarity of the frequencies in the stego image is a measure of the probability of embedding. It is performed by using the Chi-square test. The χ 2 value is given as:

$$x^2 = \sum_1^{v+1} \frac{(y_i - y_i^*)^2}{y_i^*}$$ (Provos, 2001)

where ‘v’ are the degrees of freedom, it is one less than the number of different categories in the histogram.

5. Finally, the probability, p, of embedding is given by the complement of cumulative distribution function as given below, where Γ is the Gamma function.

$$p = 1 - \int_0^{\omega^2} \frac{t^{(v-2)/2}.e^{-t/2}}{2^{v/2}.\tau(v/2)} dt$$ (Provos, 2001)

CHAPTER FIVE
USES OF STEGANOGRAPHY

5.1 Introduction

Steganography can be used anytime you want to hide data. There are many reasons to hide data but they all boil down to the desire to prevent unauthorized people from becoming aware of the existence of a message. In the business world steganography can be used to hide a secret chemical formula or plans for a new invention. Steganography can also be used for corporate espionage by sending out trade secrets without anyone at the company being any the wiser. Steganography can also be used in the non-commercial sector to hide information that someone wants to keep private. Spies have used it since the time of the Greeks to pass messages undetected. Terrorists can also use steganography to keep their communications secret and to coordinate attacks. It is exactly this potential that we will investigate in the next section.

5.2 Terrorists and Steganography

Now that we have investigated the basics of steganography we will examine what part it has had in the communication for terrorist networks and more specifically how it is linked to the activities of Osama bin Laden and the al-Qaida network. There is a general belief that some of the plans for the September 11 attacks were hidden in images and put into sports and pornographic bulletin boards.

5.2.1 Known Communications

The al-Qaida terrorist network has been known to use encryption. They receive money from Muslim sympathizers, buy computers and then go online and download encryption programs from the web. (Kelley, 2001) Here are brief accounts from USA today that describe three instances where terrorists have used some sort of encryption:

• Wadih El Hage, one of the suspects in the 1998 bombing of two U.S. embassies in East Africa, sent encrypted e-mails under various names, including "Norman" and "Abdus Sabbur," to "associates in al Qaida," according to the Oct. 25, 1998, U.S. indictment against him. Hage went on trial Monday in federal court in New York.

• Khalil Deek, an alleged terrorist arrested in Pakistan in 1999, used encrypted computer files to plot bombings in Jordan at the turn of the millennium, U.S. officials say. Authorities found Deek's computer at his Peshawar, Pakistan, home and flew it to the National Security Agency in Fort Meade, Md. Mathematicians, using supercomputers, decoded the files, enabling the FBI to foil the plot.

• Ramzi Yousef, the convicted mastermind of the World Trade Center bombing in 1993, used encrypted files to hide details of a plot to destroy 11 U.S. airliners. Philippines officials found the computer in Yousef's Manila apartment in 1995. U.S. officials broke the encryption and foiled the plot. Two of the files, FBI officials say, took more than a year to decrypt. (Kelley, 2001)

Osama bin Laden has used mobile phones, and satellite communications in the past but it is believed that he has stopped using them to make it more difficult to detect him. (Sieberg, 2001) Some experts believe that he only uses messengers now. For a military commander this would be highly ineffective as they have to be in contact with their subordinate commanders at all times, but bin Laden is considered a spiritual or inspirational leader and as a result does not have to maintain constant contact with his troops. They can operate in smaller cells.

The events that took place on September 11 were obviously very coordinated and the terrorists must have had to use some form of communication to coordinate their attacks. Since their communications were not detected, it would lead one to believe that they were using some type of encryption and/or message hiding system.

5.2.2 Steganography for Terrorists

Whether or not al-Qaida uses steganography, it would be a very effective high tech communication method. They can use bulletin boards and other public places where you can put images as cyber dead drops for stego-images. A dead drop is a place where you drop off a deliverable at some pre-determined time and place without ever meeting or directly communicating with the other party. Of course, communication will have to be initiated but after that, all communications/exchanges can be made in the manner outlined above. For covert purposes, this communication technique has two very distinct advantages over most other forms of communication. The first is that the communication is asynchronous, which means that it is simpler to implement and helps to avoid suspicion as involved parties aren't directly associated with each other. The second reason is that only one of the parties is required to know who the other is. This is especially valuable if one party is caught then they may not be able to divulge who they were dealing with, regardless of the interrogation methods used. The last point makes steganography an especially appealing method of communication to the al-Qaida network because they operate as cells and the anonymity that dead drops provide will help to avoid uncovering of the entire network even if some members are caught.

CHAPTER SIX
IMPLEMENTATION

6.1 Introduction

Our thesis can be classified as Steganography with BMP image files using a new approach on LSB technique. Because of there is no universal steganographic system, algorithms depend on the types of files. BMP is a good choice for its high steganographic capacity. More security needs extra pixels and extra capacity.

In the scope of this thesis, a program was developed to encode some information inside BMP file, and retrieve the encoded information back. This program was developed by using Visual Studio .NET technology and Visual C# programming language.

One of the important steps about our study is strengthening the security of embedding operations. A secret key, declares the receiver which pixels changed, dynamically be formed and embedded in carrier file that start position of the key put in the specific bytes in the header part of the file. Only the receiver knows starts position of the dynamically created embedded key and extracts the original message to the information in the key sequence.

In the following example the original byte sequence is on the left side as it seen. The embedded message is “0010111011” and the key that determining distances is “1102101042”. Firstly we embed the key portion to a random position and we put the starting position of the key to a specific byte area in the header parts of the file. We put the size of the key just before the key data while the embedding operations. Thus we know how many pixels we will need while getting the key data. Just after embedding the key data, we start to embed secret message. First of all we embed the size of the secret message like we did in the key data. We take the first bit of the message, ‘0’. We take the first index of the key sequence, ‘1’. It means we embed the bit ‘0’ to the byte after next byte of the original byte sequence. After we embed the

first bit, we take the next one, ‘0’. We take the next key value, ‘1’. We skip one byte and perform the embedding operation to the next byte in the original byte sequence. In the third bit of the secret message we get the bit ‘1’ and embed it to the next byte sequence in the original message to the key value in the next order, ‘0’. We keep going this operation on until embedding all the bits of the message. If we reach the end of the key data sequence of the embedding operation, we return to the first index and use the same key values again and again until the whole message embedded.

Each value in the key data place one byte. They are produced dynamically. We used two small integers for its highest and lowest index to produce the key values that the interval we decided. Thus we skip small distances in the file and use the pixels more efficiently.

While we want to decode a secret message, firstly we get the key values and then get the original data by using that key values. We know the starting point of the key data’s from the specific part in the header. We seek that position and get the two bytes of data that is the size of the key data. After we learn the size of the key data, we get the key values. Just after the last key value, we get the size value of the original data.

From this point we use the key values while getting the size values and original message that we embed. The key values give us which pixels are changed in the message. We get the changed bits and concat them in the same order and reach the original message.

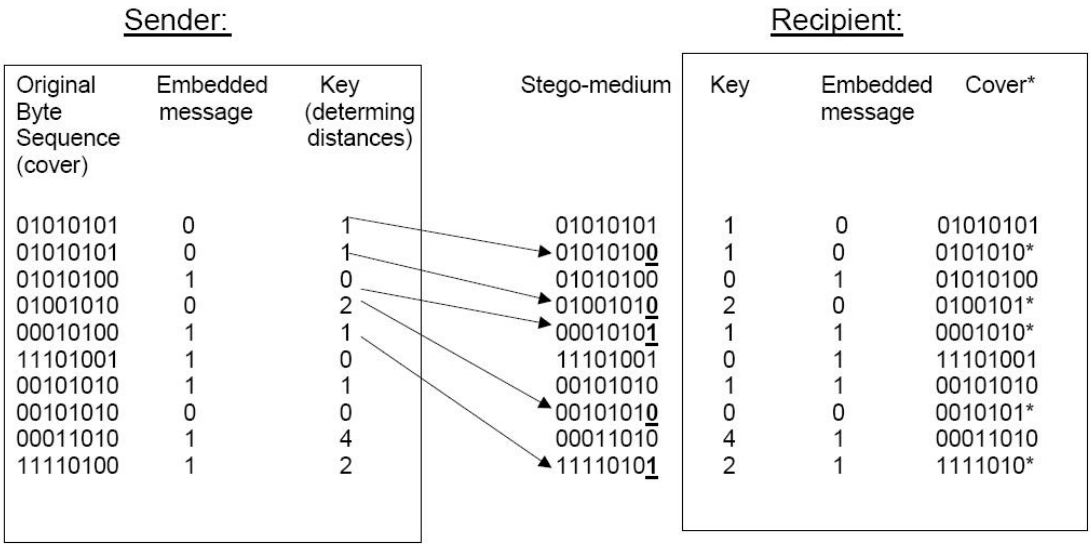
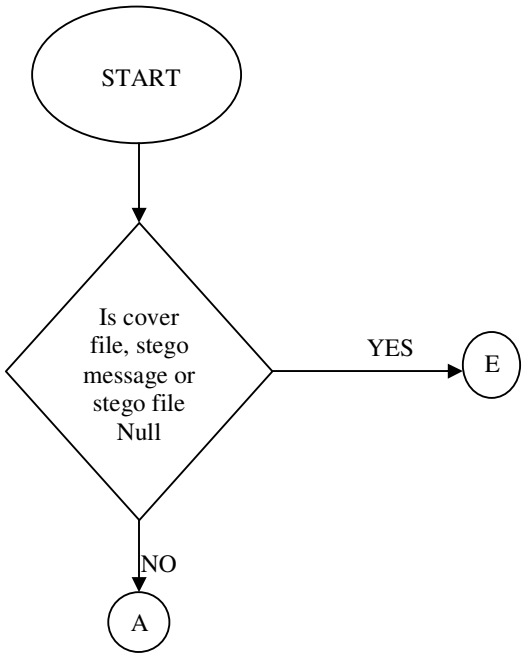
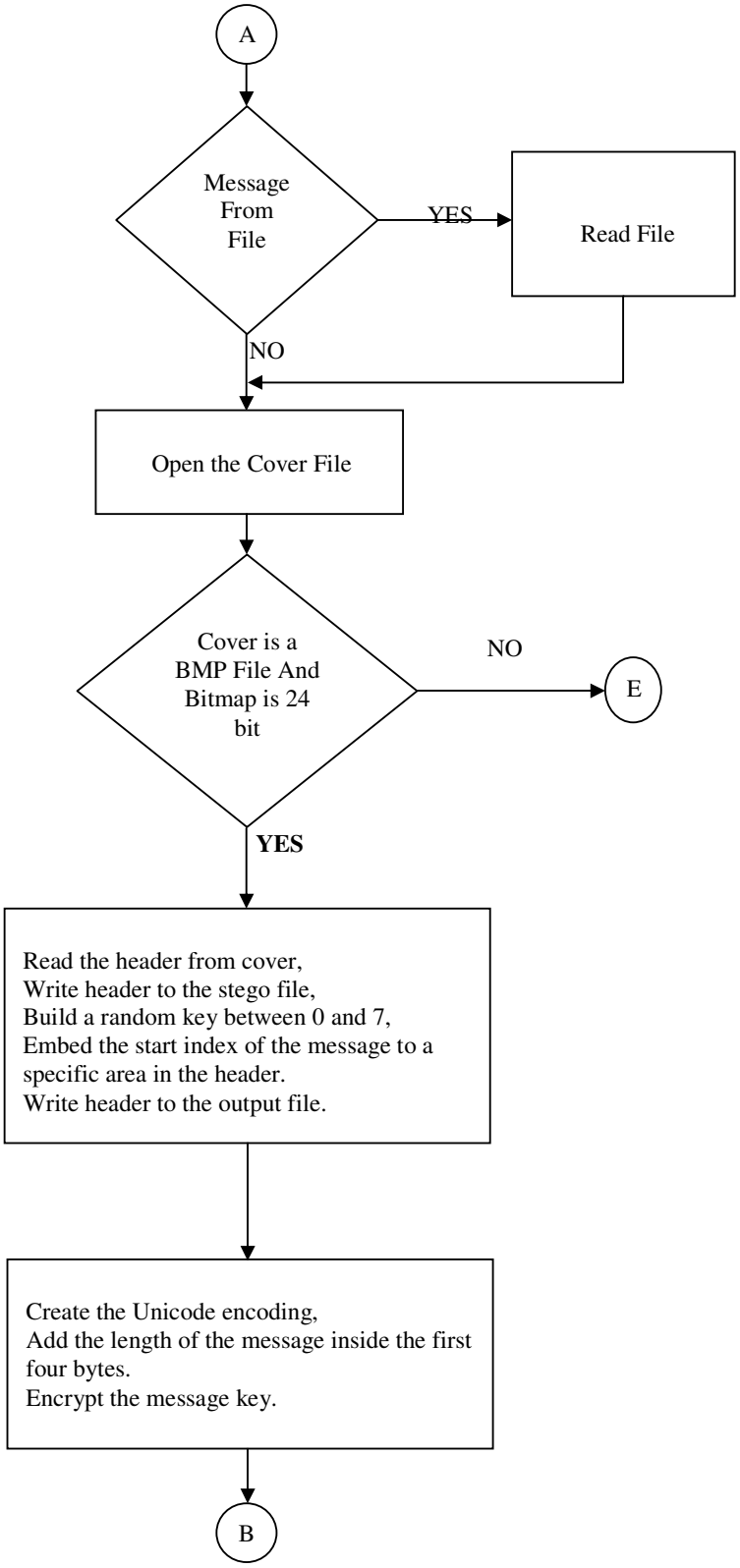
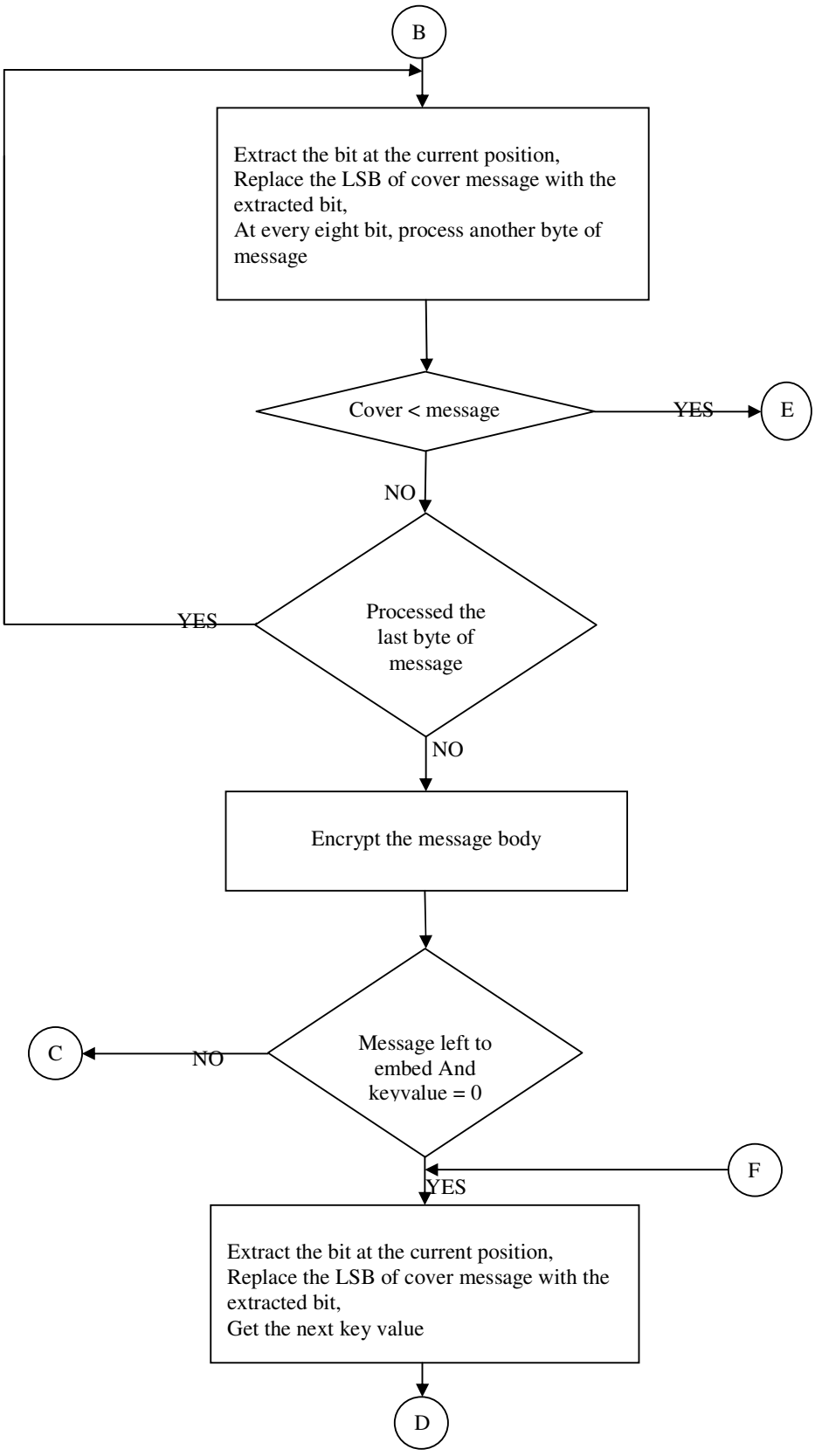


Figure 6.1 Enhanced embedding model.

In our modified model the original message and the other components like key, are encrypted and obtain a more secure system. Even someone realize that carrier file contains a hidden information he/she cannot even know where the hidden message is because of key is encrypted.




A
Message From File
YES
Read File
NO
Open the Cover File
Cover is a BMP File And Bitmap is 24 bit
NO
E
YES
Read the header from cover,
Write header to the stego file,
Build a random key between 0 and 7,
Embed the start index of the message to a specific area in the header.
Write header to the output file.
Create the Unicode encoding,
Add the length of the message inside the first four bytes.
Encrypt the message key.
B


B
Extract the bit at the current position,
Replace the LSB of cover message with the extracted bit,
At every eight bit, process another byte of message
Cover < message
YES
E
NO
Processed the last byte of message
YES
NO
Encrypt the message body
Message left to embed And keyvalue = 0
C
NO
F
YES
Extract the bit at the current position,
Replace the LSB of cover message with the extracted bit,
Get the next key value
D

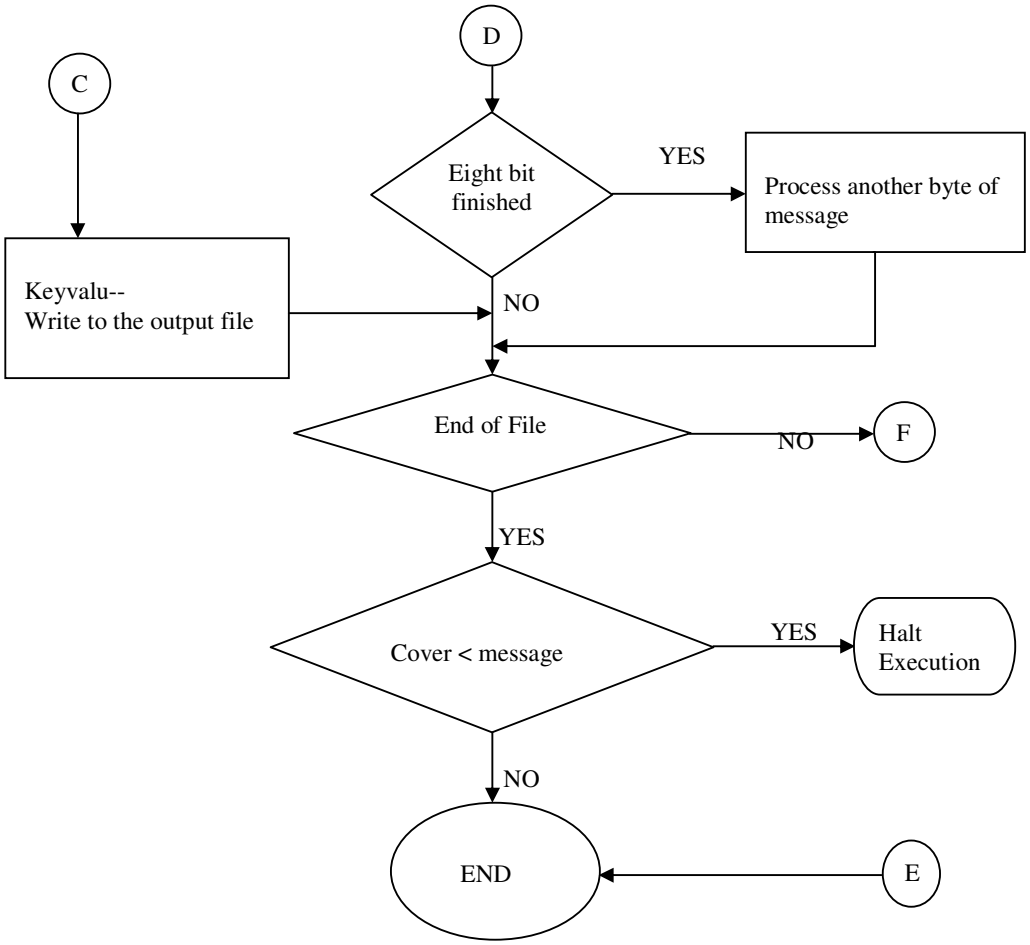
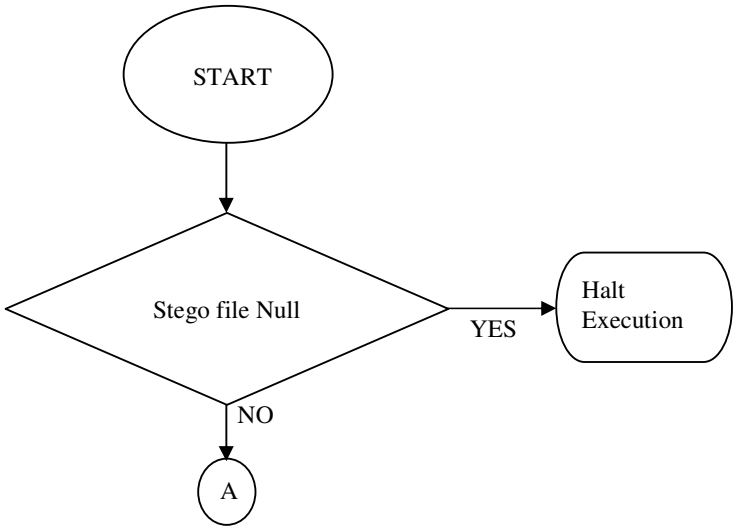
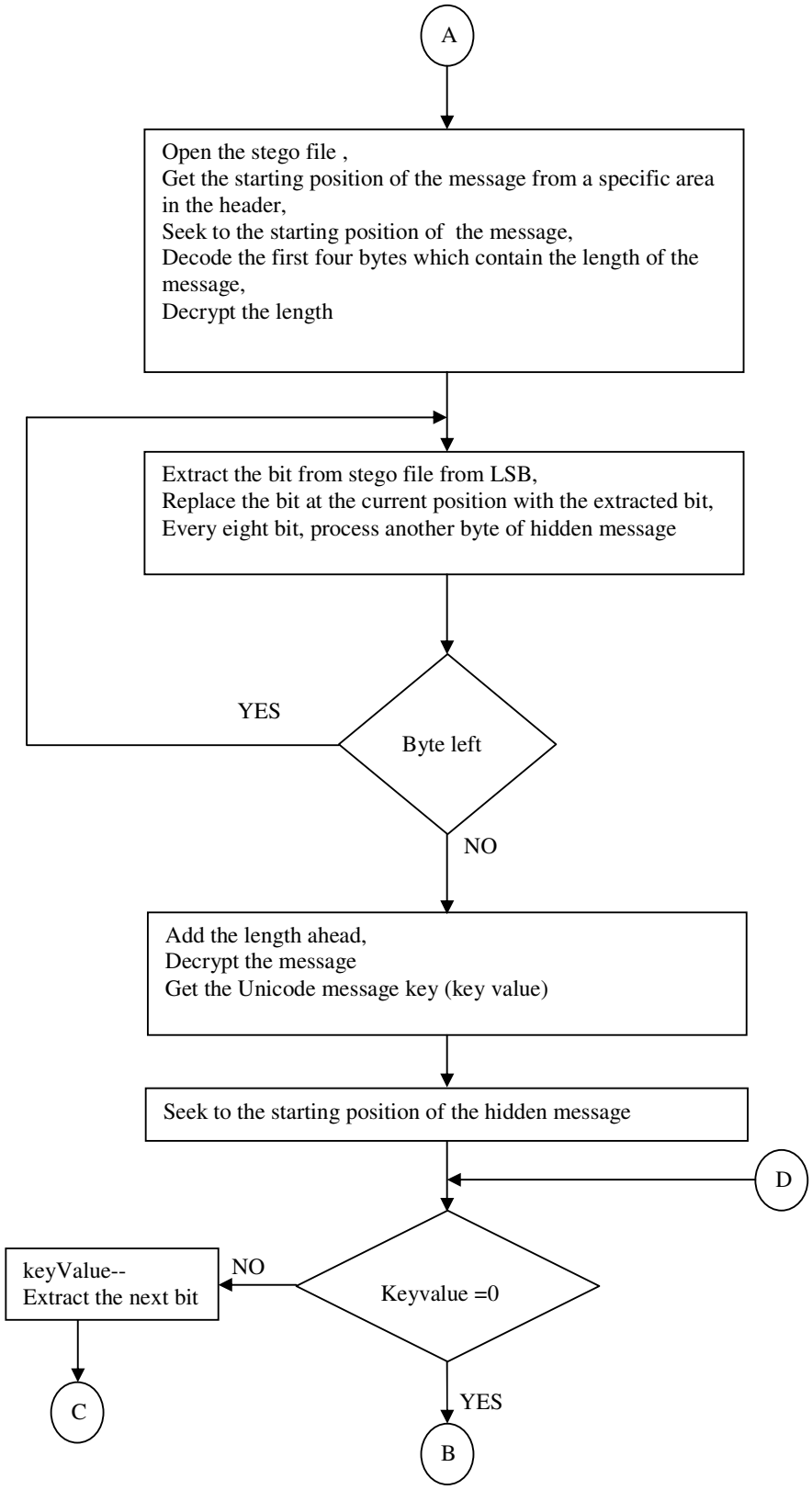


Figure 6.2 Flowchart of Encoding Algorithm




A
Open the stego file ,
Get the starting position of the message from a specific area in the header,
Seek to the starting position of the message,
Decode the first four bytes which contain the length of the message,
Decrypt the length
Extract the bit from stego file from LSB,
Replace the bit at the current position with the extracted bit,
Every eight bit, process another byte of hidden message
YES
Byte left
NO
Add the length ahead,
Decrypt the message
Get the Unicode message key (key value)
Seek to the starting position of the hidden message
D
keyValue--
Extract the next bit
NO
Keyvalue =0
C
YES
B

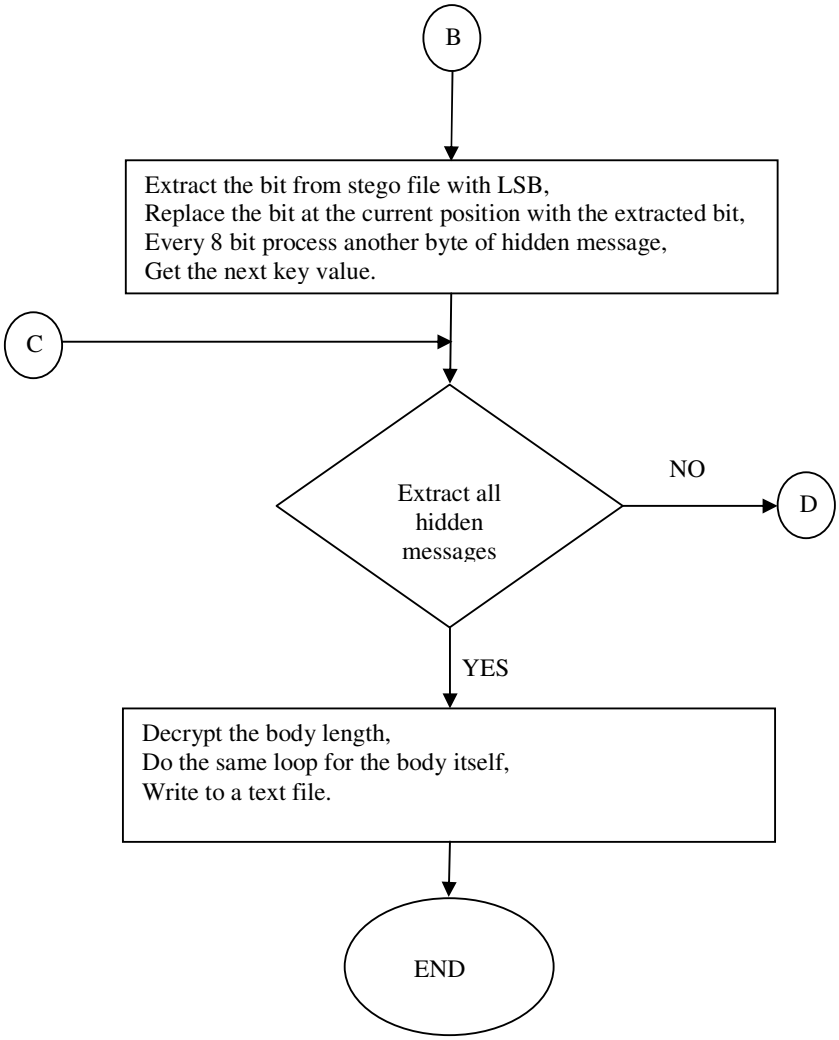


Figure 6.3 Flowchart of Decoding Algorithm

6.2 Encryption

Encryption is a frequently used operation in my thesis. Although steganography is concerned about secrecy of the message, cryptography serves the system to be more secured. Key data, original message and the header parts are all encrypted. Actually every data that is embedded to the image file is encrypted.

The encryption is based on the Triple DES algorithm. There is an assigned key data before the operation begun. The user defined password and this key data are used to construct a Triple DES key. If the user determined any password, there is a

default one in the algorithm. The triple DES key – combination of the salt data and user defined password- and the message that is going to be encrypted, will have XOR operations until all the bytes are encrypted.

6.3 Covering

Covering is the first step of embedding some information into the image files. As you see in the other chapters cover is a carrier file for the embedding data and is very important that affects the size of the data. Cover file can be chosen by the browse button at this tab page. Only the bmp extension files can be seen and chosen in the file dialog box. It is restricted because there is no universal steganography method.

Choosing cover file consists some tricky situations. The more color variety, the lower chance to detect the transformation in the carrier file with naked eye.

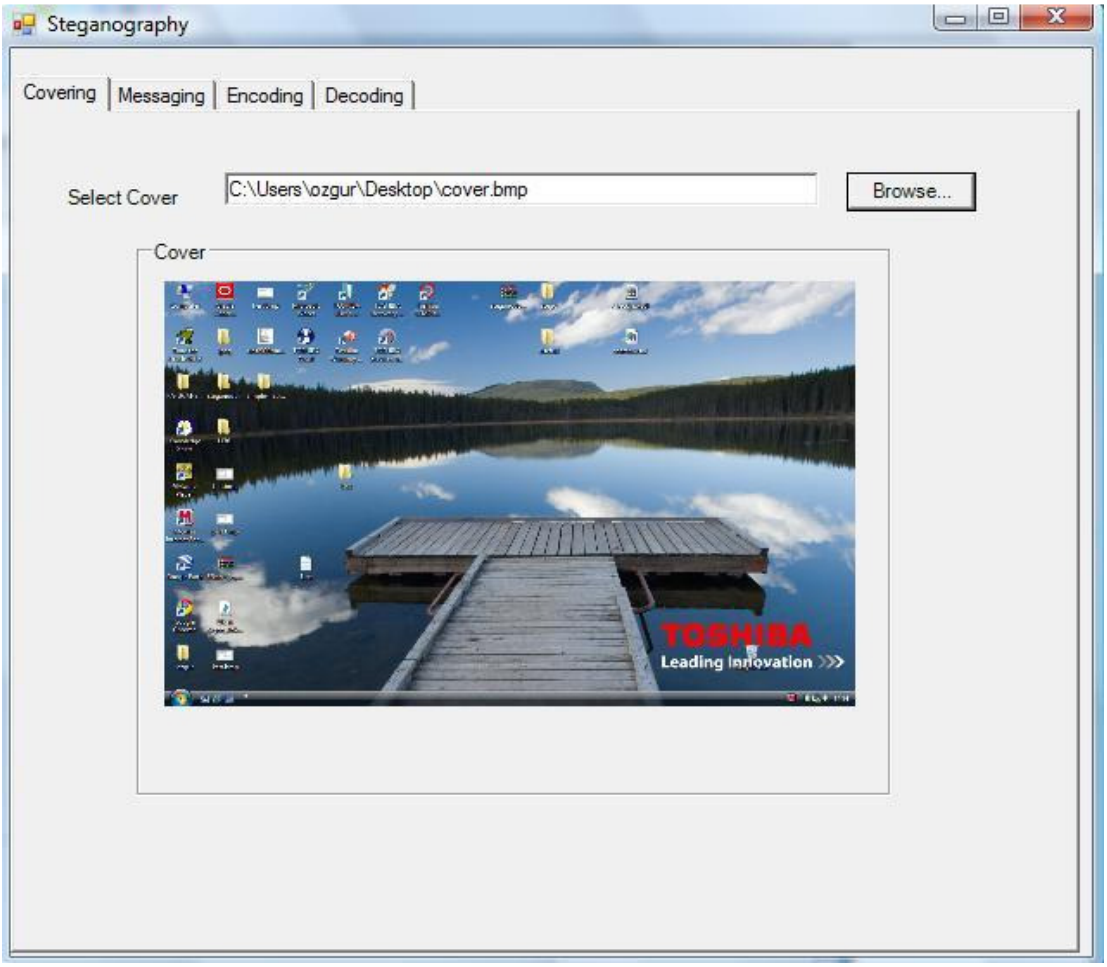


Figure 6.4 Choosing cover file

6.4 Messaging

The next step of the program is messaging part and the components about this step are in the messaging tab page. Messaging means producing the message (secret information) that embeds into the carrier file.

There are two options about messaging. One of them is shown in the figure 4.2 below. It is about writing the message directly in the field on the screen.

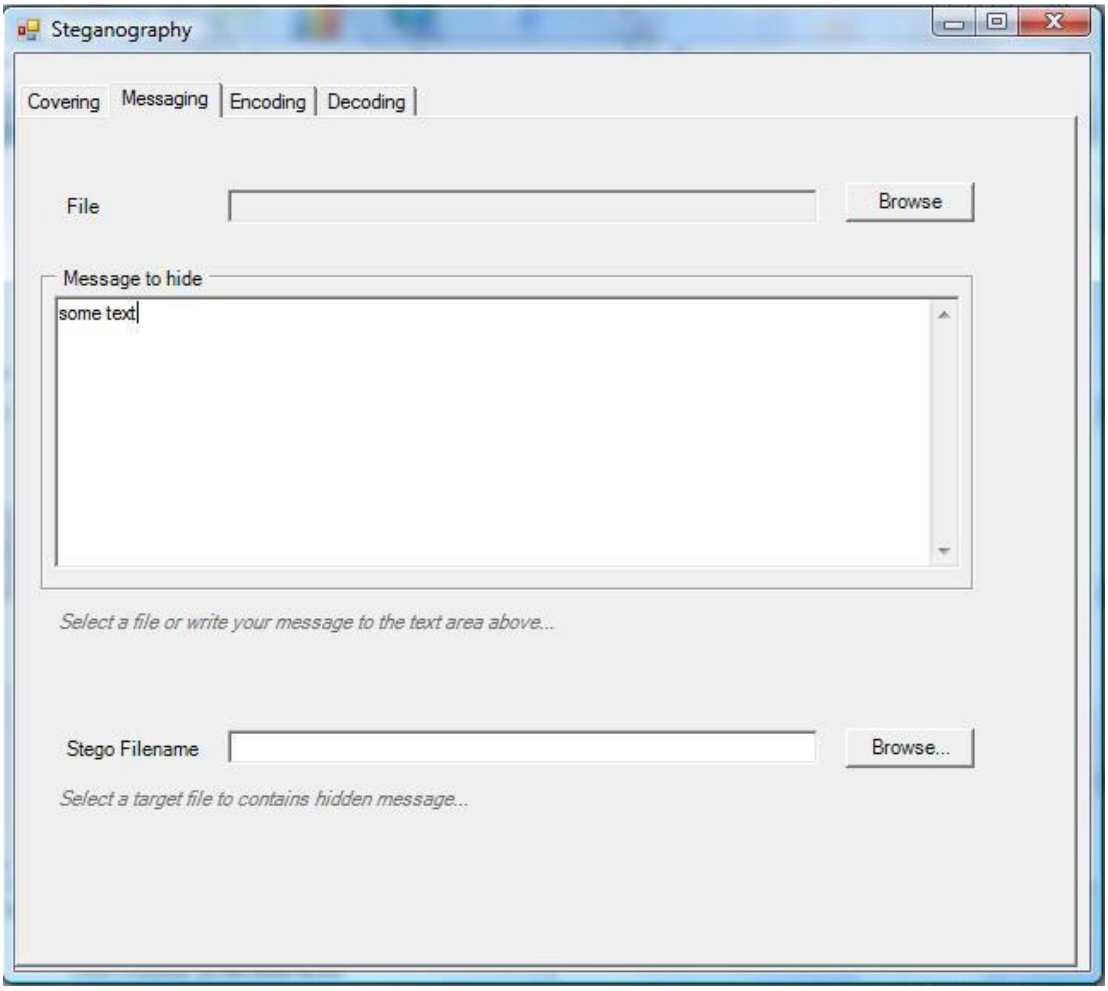


Figure 4.5 The way of creating the message to hide

The other option about creating the message is taking it from a text file. A decision cannot be made about choosing the text field or the file for taking the message to embed. So a restriction will be seen after one of them is chosen. The other choice will be disabling to the user.

The next step of the embedding operation is choosing a stego file which is a combination of carrier file and the hidden information. It will be created after the encoding operation. An existing file can be chosen or a new file can be created. The same restriction about the extension of the stego file will be the same as the other one.

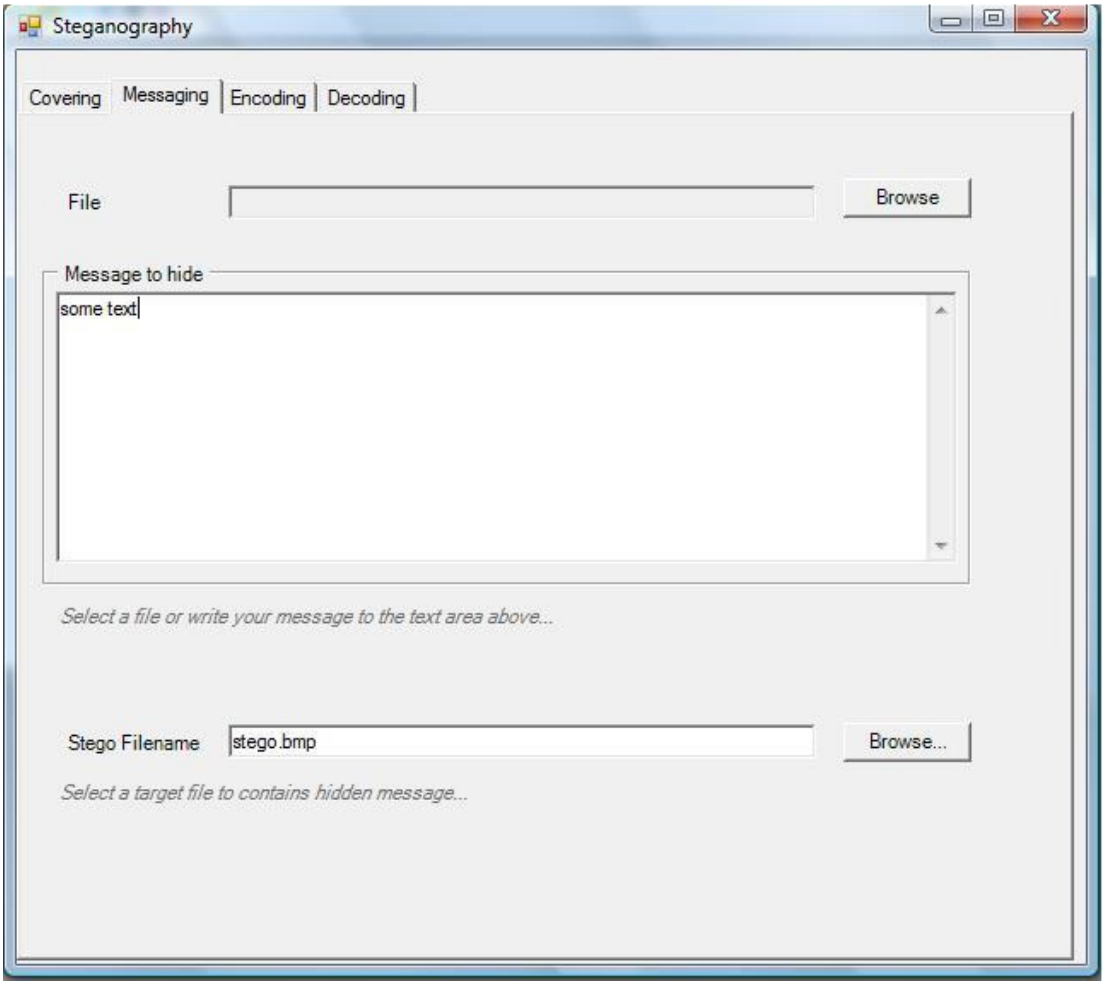


Figure 6.6 Choosing the stego file

6.5 Encoding

The next step is the core of the program. The components are located in the encoding tab page. A password field will be seen in the top of the page. This password is decided by the sender and the receiver in a session at the beginning of all operations begun. The importance of the password came from the security of the hidden information and also secrecy and importance of the hidden information. A key will be produced based on this password.

Encoding process is a bit complicated. A dynamic key sequence is created and embedding in a specific position that the sender decided. This key gives to the

receiver which bits were used to embed the message. Instead of embedding message sequentially, embedding it in a spreading way.

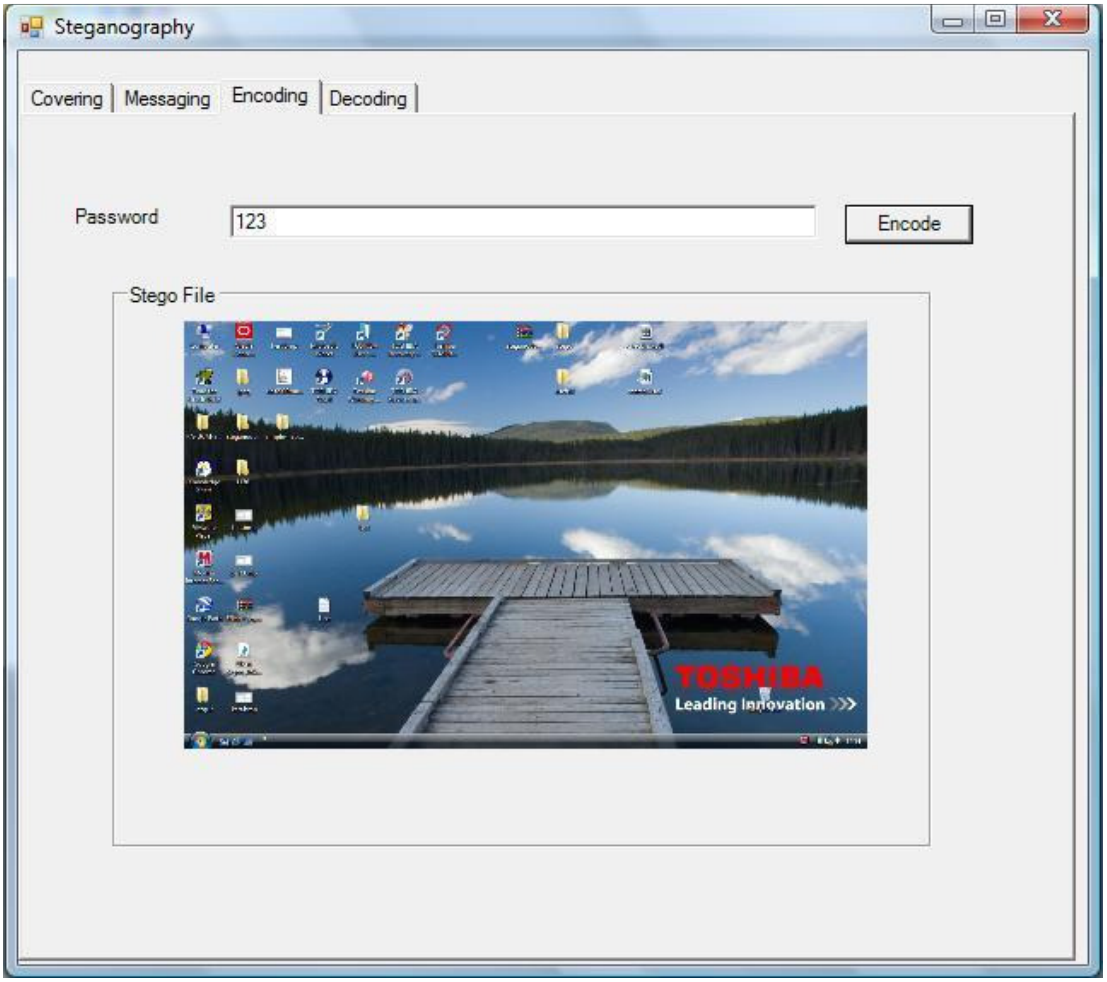


Figure 6.7 Encode operations

The point that we take into account is the size of the message and the capacity of the carrier file. If the capacity is not enough for carrying the message a warning will be displayed on the screen that the carrier file is not suitable.

6.6 Decoding

After the encoding operation there is nothing to do in the side of sender. The last tab page is about the receiver. Select the stego file that was taken from the sender.

At the final step the password that the sender used while encoding the message, is put in the password field and take the original message back in a reverse operation series.

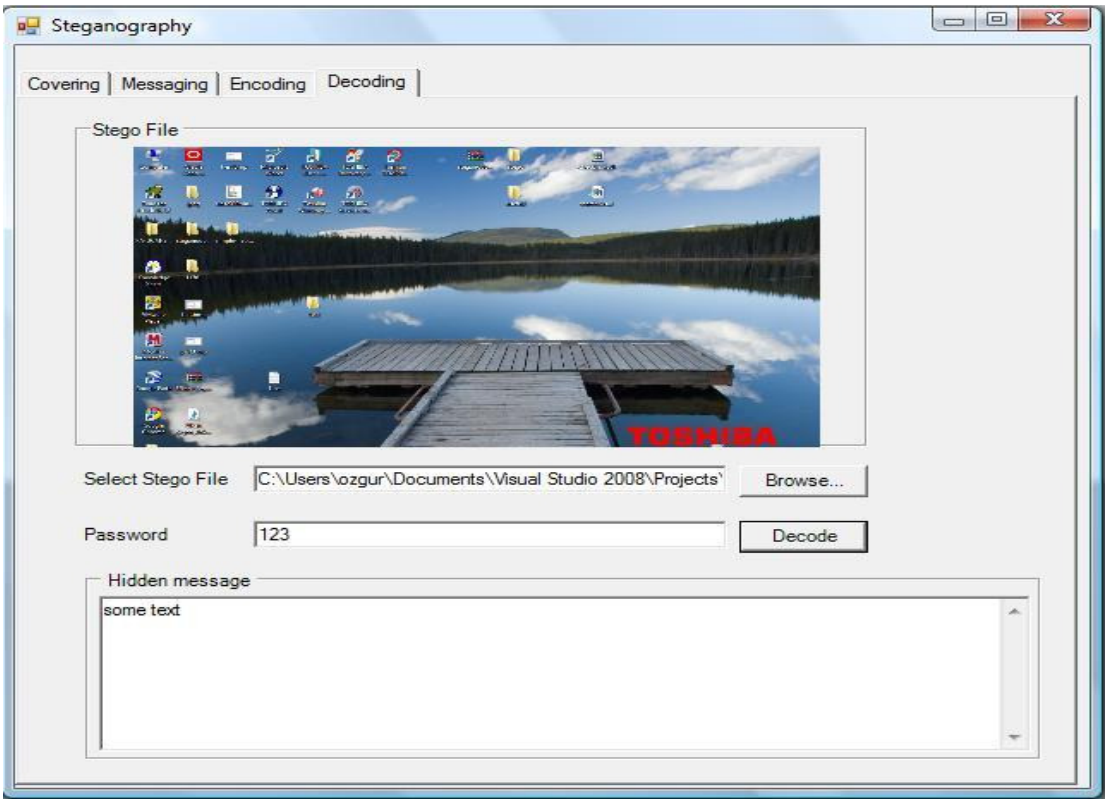


Figure 6.8 Decode operation

The difference between the original image and the stego image that contains a hidden message is shown in figure 4.6 and figure 4.7

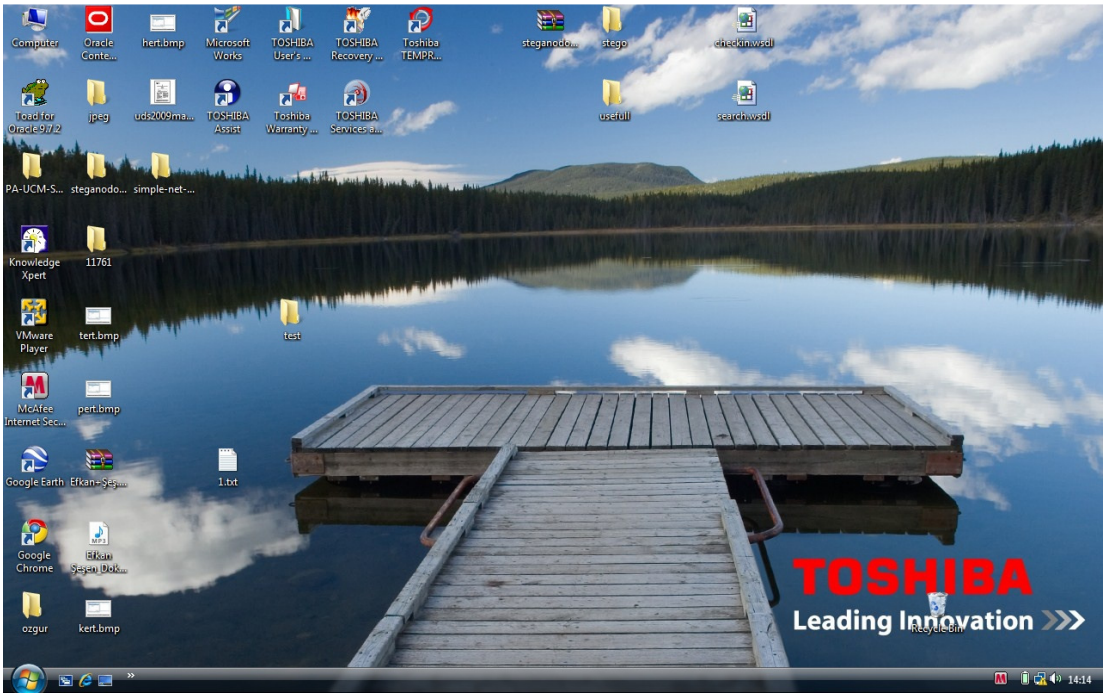


Figure 6.9 Original image

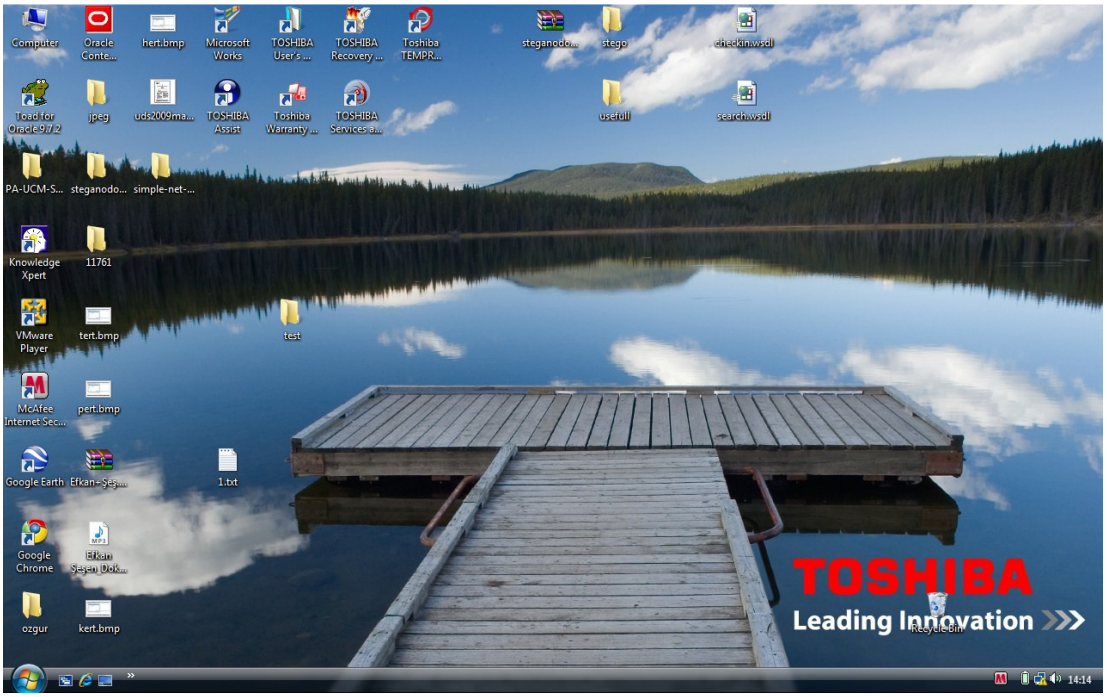


Figure 6.10 Stego image

CHAPTER SEVEN
CONCLUSION & FUTURE WORK

The 11th of September 2001 changed the perception of the threads and the methodologies of struggling with these threads. Classical security concept leaves its place to the total security perception. Computer and data security are some important two of them. Articles of newspaper claim that terrorists used steganography to communicate without anyone else's knowledge. Steganography is an unsolved problem; the reason why this study has been chosen is that there is still a little investigation on steganography over the world.

Our thesis provides an investigation of steganography and how it is implemented. Pure Steganography is not sufficient to our data be in secure. This thesis gives some solutions to handle these problems.

In conclusion to our work, modified least significant bit (LSB) technique was introduced with some extra security options for BMP files to make the system safer. Before hiding, a secret message was encrypted to get more secure form. As a result of what we implemented, we have improved the technique of embedding that hiding process is done with the arbitrary chosen pixels in BMP files. To perform spreading, pseudo random generators were used. We have shown that security is the core item in steganographic systems, and what makes our steganographic system more powerful against detection that we have already got.

Finally one can see that there exists a large selection of approaches to hiding information in images. All the major image file formats have different methods of hiding messages with different strong and weak points respectively where one technique lacks in payload capacity, the other lacks in robustness. Anyone who is making steganography takes into account these trades off and chooses the best solution for its application.

Security is a keyword for steganography and it will always play important roles in this area. So as a future work there can be a study on increasing the security. Adding some extra security options will make more valuable this study and gives the people an endless study area.

This software that we implemented can also be extended for detecting messages in BMP image files as a future work. Encryption and steganography can be implemented with multi different techniques for a company which depends on specific conditions.

REFERENCES

Anderson, R. J., & Petitcolas, F. A. P. (1998). On the limits of steganography. *IEEE journal of selected areas in communications, 16* (4), 474-481.

Bender, W., Gruhl, D., Morimoto, N., & Lu, A. (1996). Techniques for data hiding. *IBM systems journal*, *46* (3), 313-336.

Bosselaers, A., Dobbertin, H., & Preneel, B. (1997). The RIPEMD-160 cryptographic hash function. *Dr. Dobb's journal*, *22* (1), 24-28.

Chandramouli, R., Kharrazi, M., & Memon, N. (2003). Image steganography and steganalysis: Concepts and practice. *Lecture notes in computer science, 2939* (2004), 35-49.

Currie, D. L., & Irvine, C. E. (1996). *Surmounting the effects of lossy compression on steganography.* Retrieved June 28, 2009, from http://www.securiteinfo.com/ebooks/palm/irvine-stega-jpg.pdf/.

Day, B., & Knudsen J. (1998). *Image processing with java 2D.* Retrieved October 14, 2009, from www.javaworld.com/javaworld/jw-09-1998/jw-09-media_p.html/.

Dunbar, B. (2002). *Steganographic techniques and their use in an open-systems environment*. Washington, DC: SANS Institute.

Daemen, J., & Rijmen, V. (2002). Rijndael, the advanced encryption standard. *Dr. Dobb's journal* , *26* (3), 137-139.

Dobbertin, H. (1996). The status of MD5 after a recent attack. *CryptoBytes*, *2* (2), 16.

Dobbertin, H., Bosselaers, A., & Preneel, B. (2004). *The RIPEMD-160 page.* Retrieved September 09, 2009, from http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html/.

Efford, & Nick (2000). *Digital image processing a practical introduction using java* (3rd ed.). Boston: Addison-Wesley.

Ferguson, N., & Schneir, B. (2003). *Practical cryptography*. Indianapolis: Wiley Publishing.

Franz, E., & Pfitzmann, A. (2000). Steganography secure against cover-stego attacks. *Lecture notes in computer science, 1768* (1), 29-46.

Fridrich, J. (1999). A new steganographic method for palette-based images. Retrieved July 28, 2009, from www.ws.binghamton.edu/fridrich/Research/ihw99_paper1.dot/.

Fridrich, J., Goljan, M., & Du, R. (2001). Detecting LSB steganography in color and gray-scale images. *IEEE multimedia and security*, *8* (4), 22-28.

Gatliff, B. (2003). *Encrypting data with blowfish algorithm.* Retrieved September 09, 2009, from http://embedded.com/showArticle.jhtml?articleID=12800442.

Grabbe, O. (2003). The DES algorithm illustrated. *Laissez Faire city times, 2* (28), 8-30.

Johnson, N. F., & Jajodia, S. (1998). Exploring steganography: Seeing the unseen. *Computer journal, 31* (2), 26-34.

Johnson, N. F., & Sushil, J. (1998). *Steganalysis of images created using current steganography software*. Retrieved October 25, 2009, from http://www.jjtc.com/ihws98/jjgmu.html/.

Johnson N. F., Zoran D., & Sushil J. (2001). *Information hiding, and watermarking - attacks & countermeasures*. Boston: Kluwer Academic.

Hetzl, S. (2002). *A Survey of steganography*. Retrieved June 15, 2009, from http://steghide.sourceforge.net/steganography/survey/steganography.html/.

Kelley, J. (2001). *Terror groups hide behind web encryption*. Retrieved September 09, 2009, from http://www.usatoday.com/life/cyber/tech/2001-02-05-binladen.htm/.

Kessler, G. C. (1998). *Overview of cryptography*. Boston: Auerbach

Krenn, J. R. (2004). *Steganography and steganalysis*. Retrieved September 09, 2009, from http://www.krenn.nl/univ/cry/steg/article.pdf/.

Kuhn, M. (1995). *Steganography mailing list.* Retrieved September 09, 2009, from http://www.jjtc.com/Steganography/steglist.htm/.

Lai, X. (1992). *On the design and security of block ciphers, ETH series in information processing*. Konstanz: Hartung-Gorre Verlag.

Lee, Y. K., & Chen, L. H. (2000). *High capacity image steganographic model*. Retrieved April 17, 2009, from http://www.csie.mcu.edu.tw/~yklee/Publications/HCISM-IEE2000.pdf/.

Marvel, L. M., Boncelet Jr., C. G., & Retter, C. (1999). Spread spectrum steganography. *IEEE transactions on image processing*, *8* (8), 1075-1083.

Menezes, A., Oorschot, P. V., & Vanstone, S. (1996). *Handbook of applied cryptography*. Cleveland: CRC Press.

Mercury (1999). *Cryptographic algorithms.* Retrieved September 09, 2009, from http://kremlinencrypt.com/algorithms.htm#DES/.

Moerland, T. (n.d.). *Steganography and steganalysis*, *Leiden Institute of advanced computing science*. Retrieved September 09, 2009, from www.liacs.nl/home/ tmoerl/privtech.pdf/.

Murray, J. D., & William V. R. (1996). *Encyclopedia of graphics file formats* (2nd Ed.). Sebastopol, Calif: O'Reilly and Associates Inc.

Oorschot, P., & Wiener, M. (1994). Parallel collision search with application to hash functions and discrete logarithms. *Journal of cryptology, 12* (1), 1-28.

Petitcolas, F. A. P., Anderson, R. J., & Kuhn, M. G. (1999). Information hiding – A Survey. *Proceeding IEEE*, *87* (7), 1062–1078.

Provos, N. (2001). Defending against statistical steganalysis. *10th Usenix security symposium*, *10* (1), 323–335.

Provos, N., Honeyman, P. (2002). *Detecting steganographic content on the internet.* Retrieved December 12, 2009, from http://www.citi.umich.edu/u/provos/papers/detecting.pdf/.

Rivest, R. (2003). *RFC (1321).* Retrieved September 09, 2009, from http:// www.faqs.org/rfcs/rfc1321.html/.

Sağıroğlu, S., & Tunçkanat, M. (December 04, 2002). *Güvenli internet haberleşmesi için bir yazılım: TurkSteg.* Retrieved August 15, 2009, from http://www.teknoturk.org/docking/yazilar/tt000106-yazi.htm/.

Schneier, B. (1993). Description of a new variable-length key, 64-bit block cipher (Blowfish). *Fast software encryption journal*, *809* (1), 191-204.

Schneier, B. (1995). The blowfish encryption algorithm. *Dr. Dobb's journal*, *20* (9), 137.

Sieberg, D. (2001). *Bin Laden exploits technology to suit his needs*. Retrieved September 09, 2009, from http://cnn.com/2001/US/09/20/inv.terrorist.search/.

Silman, J. (2001). *Steganography and steganalysis: An Overview*. Retrieved January 09, 2009, from http://www.sans.org/reading_room/whitepapers/stenganography/steganography_and_steganalysis_an_overview_553/.

Simmons, G. J. (1984). The prisoners' problem and the subliminal channel. *Advances in cryptology*, *83* (1), 51-67.

Stallings, W. (2003). *Cryptography and network security* (3rd ed.). New Jersey: Prentice Hall Press.

Tropical, S. (2007). *Definition of international data encryption algorithm.* Retrieved September 09, 2009, from http://www.tropsoft.com/strongenc/blowfish.htm/.

Venkatraman, S., Abraham, A., & Paprzycki, M. (2004). *Significance of steganography on data security* (2nd ed.). Washington: IEEE Computer Society.

Wang, H., & Wang, S. (2004). Cyber warfare: Steganography vs. steganalysis. *Communications of the ACM, 47* (10), 76-82.

Wayner, P. (2002). *Disappearing cryptography, information hiding: steganography and watermarking* (2nd Ed.). San Francisco: Morgan Kaufmann.

Westfeld, A., & Pfitzmann, A. (1999). *Attacks on steganographic systems*. Retrieved March 17, 2009, from http://www.dia.unisa.it/~ads/corso-security/www/CORSO-0203/steganografia/LINKS%20LOCALI/Attacks.pdf/.

Wikimedia, F. I. (2009). *Cryptography - Wikipedia, the free encyclopedia*. Retrieved September 09, 2009, from http://en.wikipedia.org/wiki/Cryptography, 12/06/2004/.

Wikimedia, F. I. (2009). *Advanced encryption standard* - W*ikipedia, the free encyclopedia*. Retrieved September 09, 2009, from http://en.wikipedia.org/wiki/Advanced_Encryption_Standard/.

Wordiq. (2004). *Definition of international data encryption algorithm.* Retrieved September 09, 2009, from http://www.wordiq.com/definition/International_Data_Encryption_Algorithm/.

APPENDIX 1

BMPCOVERFILE

```
namespace Steganography
{
   public class BMPCoverFile : ICoverFile
   {
      private string fileName;
      public BMPCoverFile(string fileName)
      {
         this.fileName = fileName;
      }
      public IStegoFile CreateStegoFile(string stegoFileName, string message, string password)
      {

         // Open the cover
         FileStream inStream = new FileStream(fileName, FileMode.Open, FileAccess.Read);

         // Check whether the cover is a bmp file
         char b = (char)inStream.ReadByte();
         char m = (char)inStream.ReadByte();
         if (!(b == 'B' && m == 'M'))
            throw new Exception("The file is not a bitmap!");

         // Check whether the bitmap is 24bit
         byte[] buffer = new byte[2];
         inStream.Seek(28, 0);
         inStream.Read(buffer, 0, 2);
         Int16 nBit = BitConverter.ToInt16(buffer, 0);
```

```
if (nBit != 24)
   throw new Exception("The file is not a 24bit bitmap!");

// Read the header from the cover
int offset = 54;          // Header Size
byte[] header = new byte[offset];
inStream.Seek(0, 0);
inStream.Read(header, 0, offset);

// Write the header to the stego file
FileStream outStream = new FileStream(stegoFileName, FileMode.Create,
FileAccess.Write);

// Build a random key between 0 and 7
Random _random = new Random();
char[] _key = new char[3];
for(int i = 0; i < 3; i++) // ilk elemanı büyük ata.....
{
   _key[i] = Convert.ToChar(_random.Next(0,3));
}

//index of the message that starts to hide
//embeds four bytes to the offset of six into the header.
byte[] startIndex = BitConverter.GetBytes(0);
for (int i = 0; i < 3; i++)
{
   header[6+i] = startIndex[i];
}

outStream.Write(header, 0, offset);

// Create the unicode encoding
```

```
            UnicodeEncoding unicode = new UnicodeEncoding();

            // Add the length of the message inside the first 4 bytes
            byte[] _keyBytes = AddLengthAhead(unicode.GetBytes(_key));
            byte[] messageBytes = AddLengthAhead(unicode.GetBytes(message));

            // Encrypt the message
            _keyBytes = CryptoHelper.Encrypt(_keyBytes, password);
            messageBytes = CryptoHelper.Encrypt(messageBytes, password);

            // Encode the cover using the LSB method
            inStream.Seek(offset, 0);
            int streamIndex = LSBHelper.Encode(inStream,_keyBytes,outStream,0);
            inStream.Seek(offset + streamIndex, 0);
            LSBHelper.EncodeBody(inStream, messageBytes, _key,
outStream,_keyBytes.Length + 4);

            inStream.Close();
            outStream.Close();

            // Return a new BMPStegoFile
            return new BMPStegoFile(stegoFileName, password);
        }

        private byte[] AddLengthAhead(byte[] messageBytes)
        {
            int len = messageBytes.Length;
            byte[] bLen = BitConverter.GetBytes(len);
            byte[] ret = new byte[len + bLen.Length];
            for (int i = 0; i < bLen.Length; i++)
                ret[i] = bLen[i];
            for (int i = 0; i < messageBytes.Length; i++)
```

```
            ret[i + bLen.Length] = messageBytes[i];
        return ret;
      }
   }
}
```

APPENDIX 2

BMPSTEGOFILE

```
namespace Steganography
{
   public class BMPStegoFile : IStegoFile
   {

      private string fileName;
      private string password;
      private string hiddenMessage;

      public BMPStegoFile(string fileName, string password)
      {
         this.fileName = fileName;
         this.password = password;
      }

      // Return the hidden message
      public string HiddenMessage
      {
         get
         {
            if (hiddenMessage == null)
               ExtractHiddenMessage();
            return hiddenMessage;
```

```
        }
    }
    protected void ExtractHiddenMessage()
    {

        // Open the bmp file
        Stream inStream = new FileStream(fileName, FileMode.Open,
FileAccess.Read);

        inStream.Seek(6,0);
        byte[] startIndex = new byte[4];
        inStream.Read(startIndex, 0, 4);
        int offset = 54 + BitConverter.ToInt32(startIndex, 0);

        // Data starts at byte 54
        //int offset = 54;
        inStream.Seek(offset, 0);

        // Decode the first 4 bytes which contain
        // the length of the message, using the LSB method
        byte[] bLen = LSBHelper.Decode(inStream, 4); //Length of key

        // Decript the length
        bLen = CryptoHelper.Decrypt(bLen, password);
        int len = BitConverter.ToInt32(bLen, 0);

        // Decode the message using the LSB method
        inStream.Seek(offset + 4 * 8, 0);

        byte[] buffer;

        try
```

```
{
   buffer = LSBHelper.Decode(inStream, len);
}
catch
{
   throw new Exception("Wrong password or not a stego file!");
}

// Add the length ahead (4 bytes)
buffer = ConcatByteArray(bLen, buffer);

// Decript the message (including the first 4 bytes)
buffer = CryptoHelper.Decrypt(buffer, password);

// Extract only the message (without the first 4 bytes)
byte[] byteMessage = GetByteMessage(buffer);
// Get the hidden unicode message
UnicodeEncoding unicode = new UnicodeEncoding();
char[] charKey = unicode.GetChars(byteMessage);

//
// Body Extraction
//
int[] key = new int[charKey.Length];
//int[] key2 = new int[chr.Length];

for (int i = 0; i < charKey.Length; i++)
{
   key[i] = Convert.ToInt32(charKey[i]);
   //key2[i] = Convert.ToInt32(chr[i]);
}
```

```
        inStream.Seek(offset + ((len + 4) * 8) , 0);
        int streamIndexPosition = 0;
        int indexOfKey = 0;
        byte[] bodyLen = LSBHelper.DecodeBody(inStream, key, 4, out
streamIndexPosition,indexOfKey);
        bodyLen = CryptoHelper.Decrypt(bodyLen, password);
        int bodyLength = BitConverter.ToInt32(bodyLen, 0);

        inStream.Seek(offset + ((len + 4) * 8) + streamIndexPosition, 0);
        byte[] bodyBuffer;
        indexOfKey = 5 % key.Length;
        try
        {
           bodyBuffer = LSBHelper.DecodeBody(inStream, key, bodyLength, out
streamIndexPosition, indexOfKey);
        }
        catch
        {
           throw new Exception("Wrong password or not a stego filee!");
        }

        // Add the length ahead (4 bytes)
        bodyBuffer = ConcatByteArray(bodyLen, bodyBuffer);

        // Decript the message (including the first 4 bytes)
        bodyBuffer = CryptoHelper.Decrypt(bodyBuffer, password);

        // Extract only the message (without the first 4 bytes)
        byte[] byteMessageBody = GetByteMessage(bodyBuffer);


        hiddenMessage = unicode.GetString(byteMessageBody);
```

```
            inStream.Close();

        }

        // Concatenate two byte arrays
        private byte[] ConcatByteArray(byte[] a, byte[] b)
        {
            byte[] ret = new byte[a.Length + b.Length];
            for (int i = 0; i < a.Length; i++)
                ret[i] = a[i];
            for (int i = 0; i < b.Length; i++)
                ret[i + a.Length] = b[i];
            return ret;
        }

        // Return a the byte array in input without
        // the first 4 bytes which represent the
        // length of the message
        private byte[] GetByteMessage(byte[] buffer)
        {
            byte[] ret = new byte[buffer.Length - 4];
            for (int i = 4; i < buffer.Length; i++)
                ret[i - 4] = buffer[i];
            return ret;
        }
    }
}
```

APPENDIX 3

CRYPTOHELPER

```
namespace Steganography
{

   public class CryptoHelper
   {

      public static byte[] Encrypt(byte[] message, string password)
      {
         return EncryptDecrypt(message, password);
      }

      public static byte[] Decrypt(byte[] message, string password)
      {
         return EncryptDecrypt(message, password);
      }

      private static byte[] EncryptDecrypt(byte[] message, string password)
      {

         int passwordLength = password.Length;
         if (passwordLength == 0)
            password = "devx";

         byte[] salt = { 31, 1, 76 };
         PasswordDeriveBytes pdb = new PasswordDeriveBytes(password, salt);
         byte[] key = pdb.GetBytes(128);

         byte[] retMessage = new byte[message.Length];
         for (int i = 0; i < message.Length; i++)
```

```
            {
               int index = i % key.Length;
               retMessage[i] = (Byte)(key[index] ^ message[i]);
            }

            return retMessage;
         }
      }
}
```

APPENDIX 4
LSBHELPER

```
namespace Steganography
{
   public class LSBHelper
   {
      public static int index;
      // Create a Stream which hides the bytes contained
      // in "message" by using the LSB method
      public static int Encode(Stream inStream, byte[] message, Stream outStream, int
position)
      {
         TextWriter tw = new StreamWriter("cover.txt");
         int byteRead;
         byte byteWrite;
         int i = 0;
         int j = 0;
         int start = 0;
         byte bit;

         while ((byteRead = inStream.ReadByte()) != -1)
```

```
        {
            byteWrite = (byte)byteRead;
            // Embed the key
            if (start >= position && i < message.Length)
            {
                // Extract the bit at the j position
                bit = Bit.Extract(message[i], j++);
                tw.Write(bit);
                // Replace the LSB of byteWrite with "bit"
                Bit.Replace(ref byteWrite, 0, bit);

                // Every 8 bits process another byte of "message"
                if (j == 8) { j = 0; i++; tw.WriteLine(); }
            }

            outStream.WriteByte(byteWrite);
            start++;
            if (i == message.Length)
            {

                break;
            }

        }
        tw.Close();
        if (i < message.Length)
            throw new Exception("The cover is too small to contain the message to
hide");
        return start;
    }
```

```
public static void EncodeBody(Stream inStream, byte[] message, char[] key,
Stream outStream, int position)
{
   int byteRead;
   byte byteWrite;
   int i = 0;
   int j = 0;
   int start = 0;
   int indexOfKey = 0;
   int keyValue = Convert.ToInt32(key[indexOfKey]);
   byte bit;

   while ((byteRead = inStream.ReadByte()) != -1)
   {
      byteWrite = (byte)byteRead;
      if (i < message.Length && keyValue == 0)
      {
         // Extract the bit at the j position
         bit = Bit.Extract(message[i], j++);

         // Replace the LSB of byteWrite with "bit"
         Bit.Replace(ref byteWrite, 0, bit);

         // Every 8 bits process another byte of "message"
         if (j == 8) { j = 0; i++; }

         indexOfKey = (indexOfKey + 1) % key.Length;
         keyValue = Convert.ToInt32(key[indexOfKey]);
      }
      else
         keyValue--;
      start++;
```

```
            index = indexOfKey;
            outStream.WriteByte(byteWrite);
        }
        if (i < message.Length)
            throw new Exception("The cover is too small to contain the message to hide");
    }

    // Return the hidden bytes for a Stream
    public static byte[] Decode(Stream stream, int length)
    {

        byte[] hidden = new byte[length];
        int i = 0;
        int j = 0;
        byte bit;
        int byteRead;

        while ((byteRead = stream.ReadByte()) != -1)
        {

            // Extract the LSB of byteRead
            bit = Bit.Extract((byte)byteRead, 0);

            // Replace the bit at the j position with "bit"
            Bit.Replace(ref hidden[i], j++, bit);

            // Every 8 bits process another byte of "hidden"
            if (j == 8) { j = 0; i++; }

            // Check bytes left
            if (i == length) break;
```

```
        }

        // The hidden var contains the hidden bytes
        return hidden;

    }

    public static byte[] DecodeBody(Stream stream, int[] key, int length, out int streamIndexPosition, int indexOfKey)
    {

        byte[] hidden = new byte[length];
        int i = 0;
        int j = 0;
        byte bit;
        int byteRead;
        int indexPosition = 0; // Gövdeyi gömmeye başladığımız index
        int keyValue = key[indexOfKey];

        while ((byteRead = stream.ReadByte()) != -1)
        {
            indexPosition++;
            //
            //
            //
            if (keyValue == 0)
             {

                // Extract the LSB of byteRead
                bit = Bit.Extract((byte)byteRead, 0);

                // Replace the bit at the j position with "bit"
```

```
                Bit.Replace(ref hidden[i], j++, bit);

                // Every 8 bits process another byte of "hidden"
                if (j == 8) { j = 0; i++; }

                indexOfKey = (indexOfKey + 1) % key.Length;
                keyValue = key[indexOfKey];
            }
            else
                keyValue--;
            // Check bytes left
            if (i == length) break;
        }
        streamIndexPosition = indexPosition;
        // The hidden var contains the hidden bytes
        return hidden;
      }
   }
}
```

APPENDIX 5
ICOVERFILE

```
namespace Steganography
{
   public interface ICoverFile
   {
      IStegoFile CreateStegoFile(string stegoFileName, string message, string pwd);
   }

}
```

APPENDIX 6
ISTEGOFILE

```
namespace Steganography
{
   public interface IStegoFile
   {
      string HiddenMessage
      {
         get;
      }
   }
}
```

APPENDIX 7
BIT

```
namespace Steganography
{

   public class Bit
   {

      public static void Replace(ref byte b, int pos, byte value)
      {
         b = (byte)(value == 1 ? b | (1 << pos) : b & ~(1 << pos));
      }
      public static byte Extract(byte b, int pos)
      {
         return (byte)((b & (1 << pos)) >> pos);
      }
   }
}
```