EECS 444: PROJECT FINAL REPORT

COMPUTER SECURITY

ARUNPRASATH SHANKAR

axs918@case.edu

DATA HIDING IN AUDIO FILES USING STEGANOGRAPHY

SCOPE OF THE PROJECT:

In this project, I have tried to implement the process of Data Hiding in Audio Files using Steganography. Steganography is the art of hiding information in ways that prevent its detection. It is a method akin to covert channels and invisible inks, which add another, step in security. A message in cipher text may arouse suspicion while an invisible message is not. Currently, Steganography is frequently used to place hidden **"trademarks"** in images, music and software, a technique referred to as **"watermarking".**

Digital steganography uses a host data or message known as a "Container" or "Cover" to hide another data or message in it. The conventional way of protecting information was to use a standard symmetric or asymmetric key system in encryption. Steganography, if however used along with cryptography further enhances the security level, for example, if a message is encrypted using triple DES (EDE), which requires a 168-bit key then the message becomes quite secure as far as cryptanalytic attack is concerned. Now, if this cipher text is embedded in an image, video, voice, etc., it is even more secure. If an encrypted message is intercepted, the interceptor knows the text is an encrypted message. With Steganography, the interceptor may know the object contains a message. I have tried to implement the Password Based Encryption scheme based on RSA Laboratories PKCS#5 v2.0 standard with PBKDF1 Algorithm, which employs a hash function (MD5) using Python for this course project.

STEGANOGRAPHY PROCESS:

When performing data hiding on audio, I find that, we must certainly exploit the weakness of the **Human Auditory System** (HAS), while at the same time being aware of the extreme sensitiveness. As of today, global communication holds the key to business, personal life and almost everything. As people tend to rely on new means of communication, more and more important information is being conveyed along these new lines. In order to ensure the privacy of communication between two parties, various new methods are being developed, with

Cryptography being the mother to all those projects. However, cryptography is like a tool, it can do as well as it is programmed to do.

Also, Steganography alone is not capable of providing a sufficiently high enough level of security. In order to improve the security of the technique, I have also incorporated the encryption of data to be hidden. The true purpose of Steganography is to hide the very presence of communication by embedding a message into innocuous-looking cover objects. As long as an electronic document contains perceptually irrelevant or redundant information, it can be used as a cover for hiding secret messages. Here, covers that are digital images are stored in the Audio format. Each steganographic communication system consists of an embedding algorithm and an extraction algorithm. To accommodate a secret message, the original message, also called the **cover-audio**, is slightly modified by the embedding algorithm. As a result, the **stego-audio** is obtained. **Steganalysis** is the art of discovering hidden data in cover objects.

The ability to detect secret messages in audio is related to the **message length.** Obviously, the less information we embed into the cover-audio, the smaller the probability of introducing detectable artifacts by the embedding process. Each steganographic method has an upper bound on the **maximal safe message length** (or the bit-rate expressed in bits per pixel or sample) that tells us how many bits can be safely embedded in a given audio without introducing any statistically detectable artifacts. Determining this maximal safe bit-rate (or steganographic capacity) is a non-trivial task even for the simplest methods. The **choice of cover-audio** is important because it significantly influences the design of the stego system and its security.

AIM OF THE PROJECT:

The aim here is to come up with a technique of hiding the message in the audio file in such a way, that there would be **no perceivable changes** in the audio file after the message insertion. At the same time, if the message that is to be hidden were encrypted, the level of security would be raised to quite a satisfactory level. Now, even if the hidden message were to be discovered the person trying to get the message would only be able to lay his hands on the encrypted message with no way of being able to decrypt it.

DATA HIDING RESTRICTIONS:

There are several data hiding techniques available today. In each technique the host data type is fixed, but the embedded data type can be varied as per requirement. Data hiding technique should be capable of embedding data in a host signal with the following restrictions and features:

- The host signal should be non-objectionably degraded and the embedded data should be minimally perceptible. What that means is that the observer should not be able to notice the presence of the data even if it were perceptible.

- The embedded data should be directly encoded into the media rather than into a header or a wrapper so that the data remain intact across varying data file formats.

- The embedded data should be immune to modifications ranging from intentional and intelligent attempts at removal to anticipated manipulations e.g. channel noise, re-sampling, cropping, etc.

- Asymmetrical coding of the embedded data is desirable since the purpose of data hiding is to keep the data in the host signal but not necessarily to make the data difficult to access.

- The embedded data should be self clocking or arbitrarily re-entrant. This ensures that the embedded data can be recovered even when only fragments of information are available.

STEGANOGRAPHY IN AUDIO:

Data hiding in audio signals is especially challenging, because the Human Auditory System (HAS) operates over a wide dynamic range. The HAS perceives over a range of power greater than one billion to one and a range of frequencies greater than thousand to one. Sensitivity to additive random noise is also acute.

The perturbations in a sound file can be detected as low as one part in ten million, which is 80dB below ambient level. However there are some 'holes' available. While the HAS has a large dynamic range, it has a fairly small differential range. As a result, loud sounds tend to mask out the quieter sounds.

Additionally, the HAS is unable to perceive absolute phase, only relative phase. Finally there are some environmental distortions so common as to be ignored by the listener in most cases. We have tried to exploit these traits to our advantage in the methods discussed further while being careful to bear in mind the extreme sensitivities of the HAS.

REQUIREMENTS:

CHOICE OF PROGRAMMING LANGUAGE: **Python**

The Chilkat Python encryption library provides an advanced API for symmetric encryption, public-key encryption, digital signatures, hashing, and encoding/decoding. It also provides a framework for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers. The software also supports secure streams and sealed objects.

ALGORITHMS USED:

There are many algorithms available to achieve this. The need of algorithm is to develop concepts and a practical embedding method for Audio files that would provide high steganographic capacity without sacrificing security. The algorithm used in the project is **PBES1 (MD5) algorithm.** This algorithm is chosen because it provides more security. The three basic steps involved in the Steganography process:

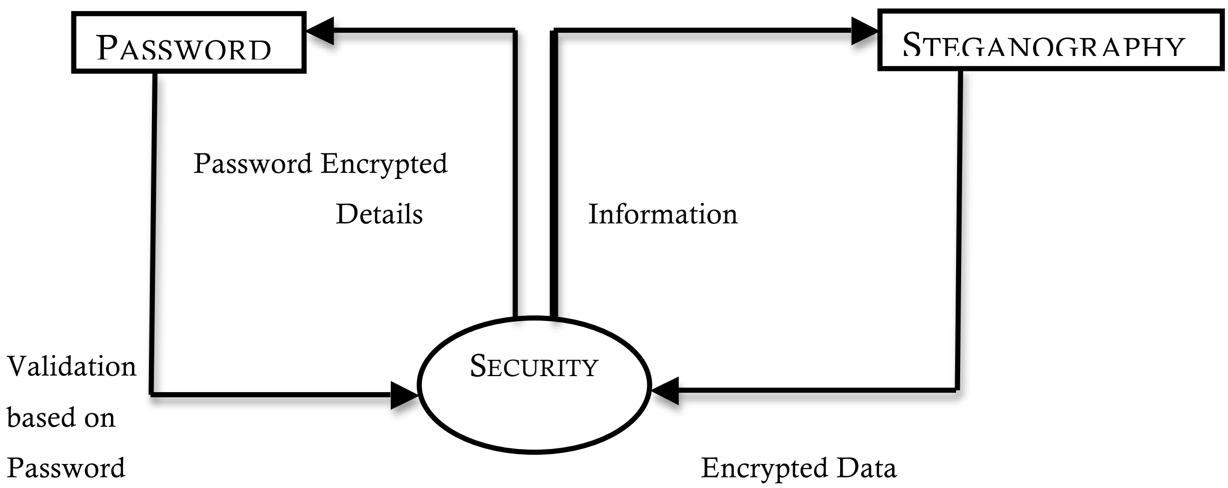
- Media Reading & Analyzing
- Message encryption and embedding into audio
- Extracting of Message from Audio and decryption

CHOICE OF AUDIO FORMAT: **.wav audio**

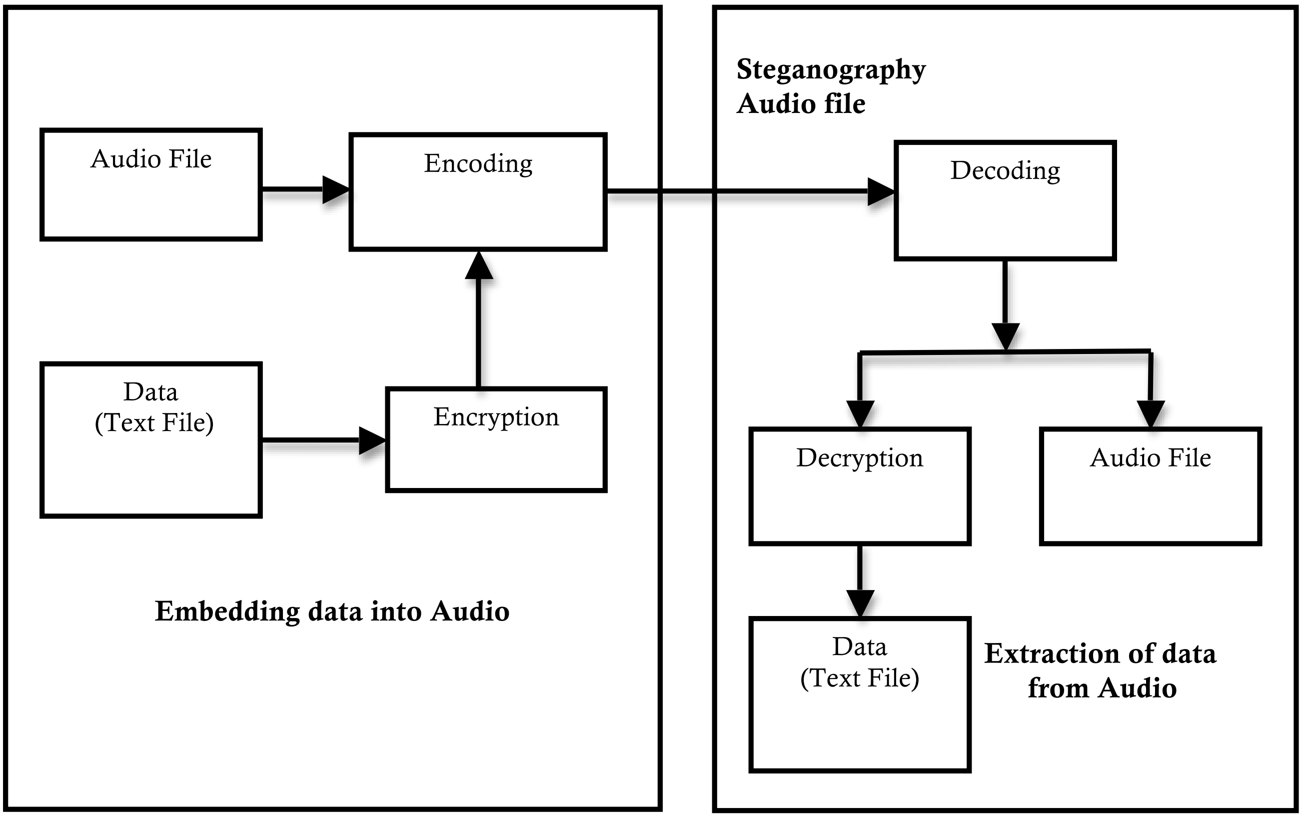
The choice of the audio also makes a very big impact on the design of a secure steganographic system. In order to demonstrate the use of Steganography techniques combined with encryption, the .wav format was chosen as the host audio file format. Python wave module is well documented and provides convenient ways to handle formatted audio data. Wav format provides the biggest space for secure Steganography, but their obvious redundancy makes them very suspicious in the first place.

Modern steganographic methods can also provide reasonable capacity without necessarily sacrificing security. MD5 algorithm is an example of a secure but high capacity Audio Steganography. Many steganographic algorithms offer a high capacity for hidden messages, but are weak against visual and statistical attacks. The algorithms used should combine both preferences, resistance against visual and statistical attacks as well as high capacity.

CONTEXT DIAGRAM:



Data Flow Diagram



Method Used To Encode Message In Audio: LSB Hiding

Least significant bit (LSB) hiding is a simple, straightforward method of hiding information in digital data. A watermark (either plaintext or some kind of cryptographically generated sequence, such as a cipher text produced by a public key cryptosystem) is generated. The cover information – in this case, an audio signal is broken down into its component samples and the least significant bit (LSB – see Figure 1) of each sample is replaced with the next bit in the watermark data. Several variants on this method exist, including combining the LSB with the watermark bit using exclusive – OR (XOR); however, all of these methods still rely on embedding the watermark information somehow in the LSB of each sample [1].

LSB hiding is a simple and fast method for embedding information in an audio signal. LSB hiding schemes provide a very high channel capacity for transmitting many kinds of data, including pseudo - random sequences produced by many cryptosystems, raw text encoded using many kinds of text encoding schemes, and even other multimedia objects, such as images and other audio signals. However, while LSB hiding presents some attractive advantages, it also presents significant disadvantages for real-world watermarking. Low-bit hiding is easily corrupted by noise as well as any kind of signal transformation or encoding, since almost any transformation will garble the watermark and render it unrecoverable without the transformation parameters. Many lossy encoders will discard information irrelevant to the human auditory system, potentially making it impossible to reconstruct the watermark from a signal that has been encoded using a lossy encoding scheme.

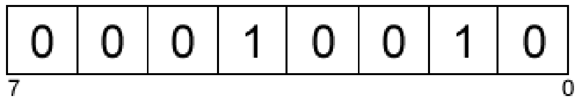


Figure 1 - A sample byte broken into its component bits. The Least Significant Bit (LSB) is the bit in the zero places, which in this case is the rightmost bit, a zero.

IMPLEMENTATION:

Formatted audio data refers to sound in any of a number of standard formats. The Python wave module distinguishes between data formats and file formats. A data format tells you how to interpret a series of bytes of "raw" sampled audio data, such as samples that have been captured from the microphone input. We might need to know, for example how many bits constitute one sample (the representation of the shortest instant of sound), and similarly we might need to know the sound's sample rate (how fast the samples are supposed to follow one another). When setting up for playback or capture, you specify the data format of the sound you are capturing or playing.

In Python wave module a data format is represented by an get_params() object, which includes the following attributes:

- Encoding technique
- Number of channels (1 for mono, 2 for stereo, etc.)
- Sample rate (number of samples per second, per channel)
- Number of bits per sample (per channel)
- Frame rate
- Frame size in bytes
- Byte order

ENCRYPTION METHOD USED:

I have used PBES1 Password-Based Encryption (PBE) according to the PKCS #5 v2.0: Password-Based Cryptography Standard (published by RSA Laboratories) which is based on the PBKDF1 function and an underlying block cipher such as RC2, DES, etc.

To start with, I first set the underlying PBE algorithm (and key length). For PBES1, the underlying algorithm must be either 56-bit DES or 64-bit RC2 (this is according to the PKCS#5 specifications at http://www.rsa.com/rsalabs/node.asp?id=2127)

Pbes Algorithm = DES

KeyLength = 56

KeyLength(168) for 3 DES Encryption (Please refer 3des.py in Project Folder)

The salt for PBKDF1 is always 8 bytes. So Encoded Salt was set to "0102030405060708", "hex". A higher iteration count makes the algorithm more computationally expensive and therefore exhaustive searches (for breaking the encryption) are more difficult. Iteration Count was set to 1024. Also a hash algorithm needs to be set for PBES1. We can use either md5 or sha1. I have used md5.

SALT AND ITERATION COUNT:

SALT:

A salt in password-based cryptography has traditionally served the purpose of producing a large set of keys corresponding to a given password, among which one is selected at random according to the salt. An individual key in the set is selected by applying a key derivation function KDF, as

$$DK = KDF (P, S)$$

where, DK is the derived key, P is the password, and S is the salt. This has two benefits:

1. It is difficult for an opponent to pre compute all the keys corresponding to a dictionary of passwords, or even the most likely keys. If the salt is 64 bits long, for instance, there will be as many as 2^{64} keys for each password. An opponent is thus limited to searching for passwords after a password-based operation has been performed and the salt is known.

2. It is unlikely that the same key will be selected twice. Again, if the salt is 64 bits long, the chance of "collision" between keys does not become significant until about 2^{32} keys have been produced. This addresses concerns about interactions between multiple uses of the same key, which may apply for some encryption and authentication techniques.

In password-based encryption, the party encrypting a message can gain assurance that these benefits are realized simply by selecting a large and sufficiently random salt when deriving a encryption key from a password. A party generating a message authentication code can gain such assurance in a similar fashion.

The party decrypting a message or verifying a message authentication code, however, cannot be sure that a salt supplied by another part has actually been generated at random. It is possible, for instance, that an opponent may have copied the salt from another password-based operation, in an attempt to exploit interactions between multiple uses of the same key. For instance, the opponent may take the salt for an encryption operation with an 80-bit key and provide it to a party as though it were for a 40-bit key. If the party performs a decryption with the resulting key, the opponent may be able to determine the 40-bit key from the result of the decryption operation, and thereby solve for half of the 80-bit. Similar attacks are possible in the case of message authentication.

To defend against such attacks, either the interactions between multiple uses of the same key should be carefully analyzed, or the salt should contain data that explicitly distinguishes between different operations. For instance, the salt might have an additional, non-random octet that specifies whether the derived key is for encryption, for message authentication, or for some other operation.

Based on this, the following is recommended for salt selection:

- If there is no concern about interactions between multiple uses of the same key with the password-based encryption and authentication techniques supported for a given password, then the salt may be generated at random. It should be at least eight octets (64 bits) long.

- Otherwise, the salt should contain data that explicitly distinguishes between different operations, in addition to a random part that is at least eight octets long. For instance, the salt could have an additional non-random octet that specifies the purpose of the derived key. Alternatively, it could be the encoding of a structure that specifies detailed information about the derived key, such as the encryption or authentication technique and a sequence number among the different keys derived from the password. The particular format of the additional data is left to the application.

ITERATION COUNT:

An iteration count has traditionally served the purpose of increasing the cost of producing keys from a password, thereby also increasing the difficulty of attack. For the methods in this document, a minimum of 1000 iterations is recommended. This will increase the cost of exhaustive search for passwords significantly, without a noticeable impact in the cost of deriving individual keys.

KEY DERIVATION FUNCTIONS

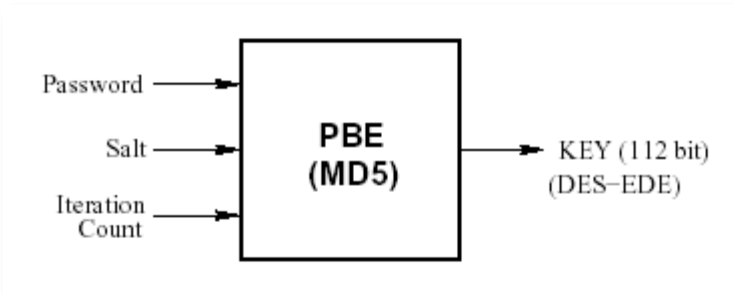
A key derivation function produces a derived key from a base key and other parameters. In a password-based key derivation function, the base key is a password and the other parameters are a salt value and an iteration count.

A typical application of the key derivation functions defined here might include the following steps:

- Select a salt S and an iteration count c, as outlined in Section 0.
- Select a length in octets for the derived key, dkLen.
- Apply the key derivation function to the password, the salt, the iteration count and the key length to produce a derived key.
- Output the derived key.

Since a password is not directly applicable as a key to any conventional cryptosystem, however, some processing of the password is required to perform cryptographic operations with it. Moreover, as passwords are often chosen from a relatively small space, special care is required in that processing to defend against search attacks. A general approach to password-based cryptography, for the protection of password tables, is to combine a password with a salt to produce a key. The salt can be viewed as an index into a large set of keys derived from the password, and need not be kept secret. Although it may be possible for an opponent to construct a table of possible passwords, constructing a table of possible keys will be difficult, since there will be many possible keys for each password. An opponent will thus be limited to searching through passwords separately for each salt.

Another approach to password-based cryptography is to construct key derivation techniques that are relatively expensive, thereby increase the cost of exhaustive search. One way to do this is to include an iteration count in the key derivation technique, indicating how many times to iterate some underlying function by which keys are derived. A modest number of iterations say 1000, is not likely to be a burden for legitimate parties when computing a key, but will be a significant burden of opponents.



Key Derivation Function (KDF) in PBE

Salt and iteration count formed the basis for password-based encryption in PKCS #5 v2.0. The PBE schemes here are based on, underlying conventional encryption schemes (for example, in this implementation, triple DES-EDE with two keys in CBC mode), where the key for the conventional scheme is derived from the password.

A salt in password-based cryptography has traditionally served the purpose of producing a large set of keys corresponding to a given password, among which one is selected at random according to the salt. An individual key in the set is selected by applying a key derivation function KDF, as:

$$DK = KDF (P, S)$$

where DK is the derived key, P is the password and S is the salt.

With this scheme, it is difficult for an opponent to pre-compute all the keys corresponding to a dictionary of passwords, or even the most likely keys. If the salt is 64 bits long, for instance, there will be as many as 264 keys for each password. It is unlikely that the same key will be selected twice. Again, if the salt is 64 bits long, the chance of "collision" between keys does not become significant until about 232 keys have been produced.

An iteration count has traditionally served the purpose of increasing the cost of producing keys from a password, thereby also increasing the difficulty of attack. Of the two functions defined in [3], we chose PBKDF1, which employs a hash function, in this case, MD5.

PBKDF1 ALGORITHM (see [3])

PBKDF1 (P, S, c, dkLen)

Options:	Hash	underlying hash function
Input:	P	password, an octet string
	S	salt, an eight-octet string
	C	iteration count, a positive integer
	dkLen	intended length in octets of derived key, a positive integer, at most16 for MD2 or MD5 and 20 for SHA-1

Output: DK derived key, a dkLen-octet string

STEPS:

1. If dkLen > 16 for MD2 and MD5, or dkLen >20 for SHA-1, output “derived key too long” and stop.
2. Apply the underlying hash function for c iterations to the concatenation of the password P and the salt S, then extract the first dkLen octets to produce a derived key DK:

   T1 = Hash(P||S),

   T2 = Hash(T1),

   …

   Tc = Hash(Tc-1),

   DK = Tc <0..dkLen-1>.
3. Output the derived key DK.

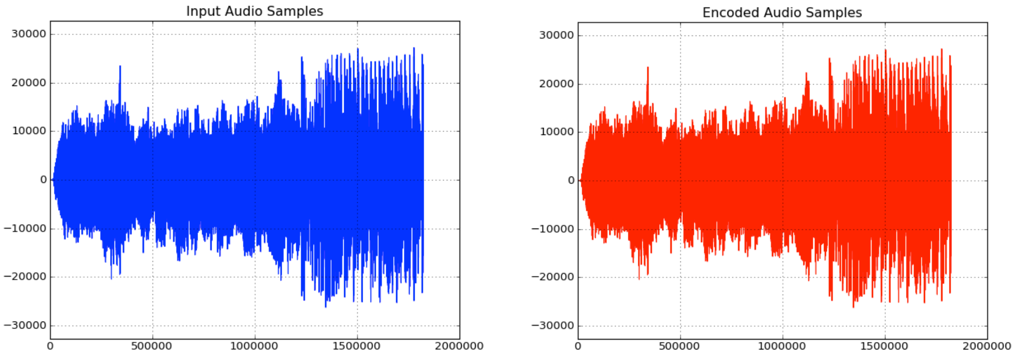
STRENGTH AND WEAKNESS:

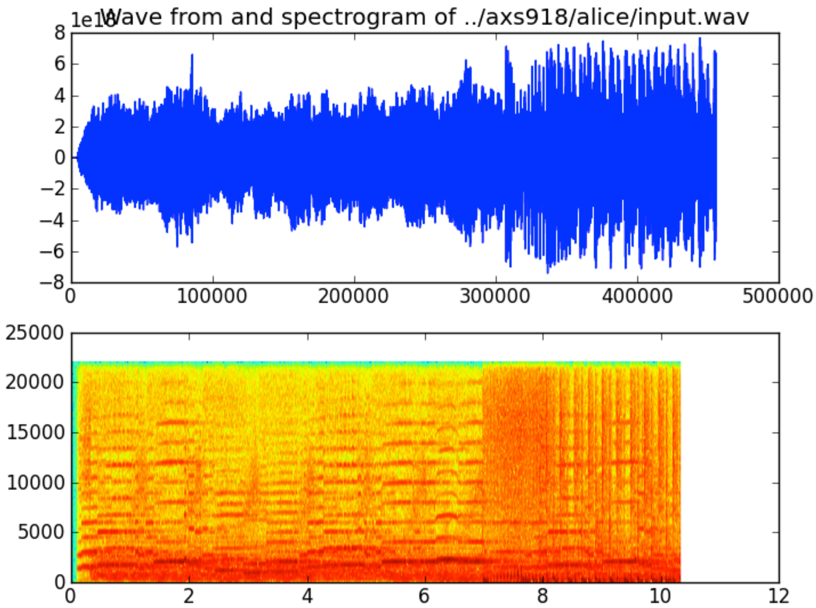
The major disadvantage of low-bit encoding is, its poor immunity to manipulation. Encoded information can be destroyed by channel noise, re-sampling etc., unless it is encoded with redundancy techniques. In order to be robust, these techniques reduce the data rate, often by one to two orders of magnitude. In practice this method is useful in closed, digital environments. Other data hiding techniques such as Phase encoding, Spread spectrum and Echo hiding have better immunity to manipulation.

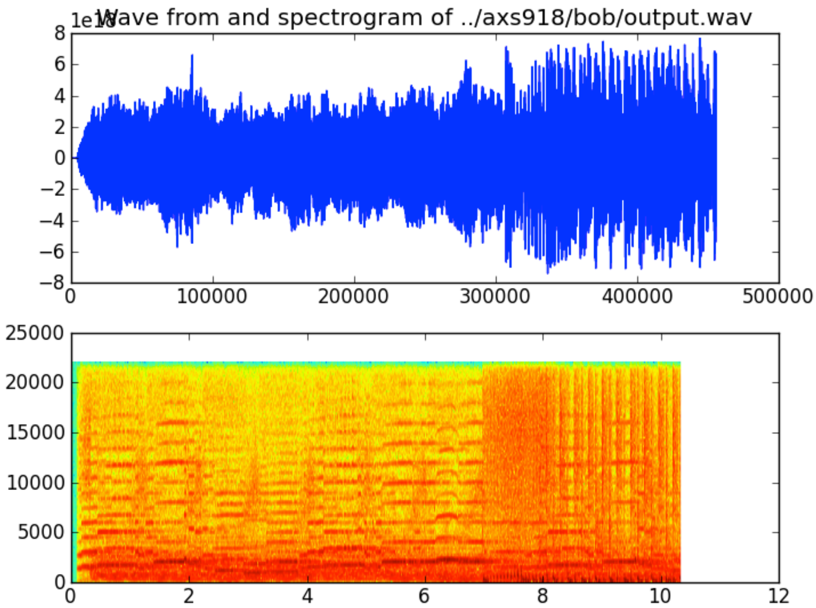
Steganalysis focuses on two aspects: detection of embedded message, and destruction of embedded message. In our implementation we noticed that the detection process is extremely difficult since has a fairly small differential range.

Examples of Steganography attacks are: Stego-only attack. Host-stego attack (known host signal attack), and chosen message attack. Trivially, our implementation is strong against those types of attacks since the attacker is unaware of the encryption algorithm and also the parameters of the algorithm. There is no secure stegosystem if the attacker knows both host medium, and stego medium. The difference reveals that Steganography was used. But the encryption adds another level of security to the conventional stegosystem.

It is necessary to maintain the same salt and iteration count to generate the same key in both encryption and decryption process. This reduces the key space in a brute force attack. But the salt, which is compiled with the source code itself, makes it harder to guess. Also the Triple DES (EDE), used here, is regarded as a better alternative to conventional DES.


Input Audio Samples
Encoded Audio Samples


Wave from and spectrogram of ../axs918/alice/input.wav


Wave from and spectrogram of ../axs918/bob/output.wav

PBES1

RESULTS

Secret Text:

Day had broken cold and gray, exceedingly cold and gray, when the man turned aside from the main Yukon trail and climbed the high earth-bank, where a dim and little-travelled trail led eastward through the fat spruce timberland. It was a steep bank, and he paused for breath at the top, excusing the act to himself by looking at his watch. It was nine o'clock. There was no sun nor hint of sun, though there was not a cloud in the sky. It was a clear day, and yet there seemed an intangible pall over the face of things, a subtle gloom that made the day dark, and that was due to the absence of sun. This fact did not worry the man. He was used to the lack of sun. It had been days since he had seen the sun, and he knew that a few more days must pass before that cheerful orb, due south, would just peep above the sky-line and dip immediately from view.The man flung a look back along the way he had come. The Yukon lay a mile wide and hidden under three feet of ice. On top of this ice were as many feet of snow.

Encrypted Secret Text using PBES1:

3FC091AEBE59E09BAE28CE2CF24A24FDC9C0DB2C8E84D7905081F96C9E8DC69BBF4FFD8BAF5253B7036BAF
3D13361E486B58E71AF97BB5738F39251E34FD51594A9FFA30B2738C5479B2238C70763D6F2A57AD088981CCEA
401F4A609BB691ECCFC269E7D3E01DD7E5A6F3D3F0BD48662ACF2B1ECDB9AFEEA6018C0E8A260F867944570
FD268E16B2058BF7F48F17876DE23AE93AA8E47574DEADA5CC853528A7B22DBAD18B5534A69CE07F44ACAC
5EDCEAA8E4F1DBD00A300913BD33BAD9F21F71F5BC4FFAD4280ACD4C66F33D874B2F744BDC677EFB8B679
80EE3A2037FC6FA3BBC6A82B8DF8BDBDED0A9D41D00B3F353BF13721BA336DB89CEE091AF709E6F43F801B
CA62005AB5AAC1D7B151F40D52AA94AEFB880AE9AD5D3C9F4C53F356BED8E1CD368EC9B8E33C1364D52427
C8643343B5519FDFC10C93BE51DCEB748D714C4EE989030EAE9F1F71A0EB8432D6EA75EF7673B9199D81457835
E43C37F046EF4CB3ECF4BCA1B979A0D8EF1545E4D8D35862090B703F80CF1E303CE0BBC5953AD024206D2C37
2CEE27DCF3D1B1D6BEFD87C09710CA54DE5F7A00B4FD1B42FA5C293DEF8C55B8EEEF95577EAAC0845A375
AA8C0D21F61A52C677B671C3C037070DC9BCEAE1EEB1CEE50B7057F623A4B3D9B368BB23C00310519E57EAA5
9FFEC0E6BAA960D4E9E96530B0A4F449D8534EC26F97F671950604C27274FFA00D8C3BDE150859FC2D31A172D
AF711A247E4BB801225FE275E3ABD2FCE6C83E5C4E202667C2B1592E292EDF915B4D9DA0471D542E274B0C79C
9A1A90D624B115BD427AFF25C28228F574CB0B4D8C19CA875E90EE52E44352A1EF94D64E388C4048D40C6D9D
6510553A3ED524A06A5B4475AC0DD2E27233EB51A6155BBD269AB28E0DCDEE2462B695744F969748922C693A0
225C5B321D05459AF969D6CB59681F583DA79432286BDF6CB12301A42EC596429327B9F33501132125C8F3467298
C37A698ADD458773C7C30BFCBF3A7AC0C0FB53DD3886E8881745D54623B6C72104CCA51204A2ED028DC75C0
396C3417A28E60A2C68E0F6E5C06E621A7C3F2A44246419878082C2B829CAC51ACA2F706B5BFC23A5EC1868459
64E08A1E0A698BDAC335C172D4A10AF4FEFA35C9BF9E4B2E9B4CE0BFEBB654E64803572CE535CDCBDCC22E
509E5676286CDA34F330E0ED5553136B556968399D7B5951B6EEF00528117D19B04B4165FF171626F6191508BCBC
F9F9B7774100EF35CCD58F85E8C90D9C8803DE32244B25D790FF39EB62B7513ECC6CAA39F5B64C4E9223740181
C177D4920CBD0246545E2660954057BDC47A57AAB43931E4F5EED50EFE5436B8DA17DF2C71A0592125C3EBC00
D49978F063CA4FF9F34D3BD64D5BDCB0909E040E71CFFADAB11B851D493E753F807BA1085B89866D1B8FFDE
3FA841

Encrypted Secret Text using 3 DES:

7A1C2DA2FB8F3F7CC56C3F28E359D89D2E8934D8F15634F543DF5A4674AE83A5BFC130A7B72F9635A6DE0983
0A95617182A16F6023F7D0EB212802B58E777C83A50000D30EED1C7A9FEB7DD57FB3BF91B6623C9246552391CC
28E36522C5484C7DAA2D602E2BA1A57681D2727592B22DF919228B45B829FE348B0C6011E098BBA36CC4BA4D5
3AC4DA3CE093E58314A194CB1EA6A0FBB4CD6B708ECDD606FDE67A38669F4501E2E2D40DB9BC552C21B7103
24835737F5702EB093AA045A01C25DA11A9ADE8836EE160CC8F82EBB29A4EDF6BE3693004CC292A4EC0231F07
644CD7F7B68BEFC66633BDD381ABEA27630F1B8E41AC77C7CD9F99A583398C7BCA5276F2164A3B401ECB49C5
3D6C70791F40568EEF6EE9ACABC53DB04F17707D28A1FD05284362132C0E74E0D1B759C75D6F3C6A72005AC77
0E9ADCD3AB5C71A0102D9E8965846C3C37B1691CDAF0D9F22AAF57E8236A19D9E7BC0ACE44F8EAEDA27B2
A2B428BDF2A1B0F184EC50A34510DC857F54AAB8F1DBB24BAFBD38AE4374174C225495E7A56D8957789E4AC
CCE61F4BAC65DB8B004A27F93E29C841B2B6372ACF0C57E132F4C5C88849F9E35DE5851E590B774090303E2F32
A1AC81FA3721B80B4FB72F02A6E9BBCF540E13DCC8FAB0690A1C146922A2823FC8913EC62B84BA0FA9BFA186
5807A507C90C33AE329BBF2428D4927223365065C2403B638CC4AB0E0D6B1B2CB87460A60F715FD44F063DBF45
C25657012157BE2C3123ECF070DC1C80D52B5030636892D9B4A8B7E4C3A31857CAFCD41F9C59F64ED4F34804B
36C3E3F3D6AE471C795212C1433CFD145981DB4ED25058C18EF04B7778752A83BF25A54C6F37EA83E365044F77
015CE0E0D4F2320F070C4E633FEBA4C35A47C9E64356FB3842CA11ABED9B68DCAA2811C32CE3F36266AA581
C76288D5B1E05A488AD18CD83B539B0A1EAA73F05739E5C622C772F13ECCD24381212CF76543CA584109067787
7F69E4299398E0B5310D061AE36770FEFE895473A1199D41D9970512170D74B1F6F49776BE818F27DB63ECFBCB5
6B82552DE883231DAC9DAEF33C5F1296D2D0B967AA67F359EE3962D256B838D84078B0C66D60EB1050D83B50B
02E630C36C417BD82D03E65CE9A5DD8B1408A3A03BD86CBFEEF467584B6BF2F4978B6567DDC9D9002D7D5B4
44655C0E3E0E48F6A47386713B20763410C284420E3735F105F313BDC95F381A1A460A776AACDC38D4628F668F6
17AC61C6F8C762931F4EDED9A01A07C3C16C1F31FDDFFEB7CDA087E64F870D90B9DEC8573CCE7C2AA5B2A
46C9BA24CD8BDA960AFB4DB78766FC8557F7A5DC898C8976163D93966D5801FF29C998967C9134DD01B268879
1820F49C7B8A1C4AF89BD4F2FBDA806D87207802CAFF31DED7528DF1362DCC3D196848D7495E83C0F852

Full Results in Project Folder

CONCLUSION:

Steganography transmits secrets through apparently innocuous covers in an effort to conceal the existence of a secret. Audio file Steganography and its derivatives are growing in use and application. In areas where cryptography and strong encryption are being outlawed, citizens are looking at Steganography to circumvent such policies and pass messages covertly.

Although the algorithm presented is a simple one and not without its drawbacks, it represents a significant improvement over simplistic steganographic algorithms that do not use keys. By using this algorithm, two parties can be communicated with a fairly high level of confidence about the communication not being detected.

REFERENCES:

[1] Kh. Manglem Singh, S. Birendra Singh and L. Shyam Sundar Singh, “Hiding Encrypted Message in the Features of Images”, IJCSNS, VOL. 7, No.4, April 2007

[2] Nameer N. EL-Emam, “Hiding a large amount of data with high security using steganography algorithm”, Journal of Computer Science, Page(s): 223 – 232, April 2007.

[3] P. Dutta, D. Bhattacharyya, and T. Kim, “Data Hiding in Audio Signal: A Review,” in International Journal of Database Theory and Application, vol. 2, no. 2, June 2009.

[4] Cheddad A, Condell J, Curran K, McKevitt P (2010) A Hash-based Image Encryption Algorithm. Opt. Comm. Elsevier Science. 283(6):879–893

[5] P. MacKenzie, S. Patel, and R. Swaminathan. Password-Authenticated Key Exchange Based on RSA. Intl. J. Information Security 9(6): 387 - 410, 2010