Audio File Encryption

by

Rhema Rosychuk

Department of Computer Science

Submitted for COSC 4235 in the Honors Bachelor of Science: Computer Science Degree

Algoma University College

Sault Ste. Marie, Ontario

April 2007

Abstract

Availability of digital media has recently increased exponentially due to the internet. Media is transferred via the internet at rates which were previously unheard of. While this decreases the time and man power required to complete a transaction, it also allows files to be illegally distributed at the same rate. This thesis paper covers the topic of Encryption and its application to audio files. Encryption has become crucial in online transactions due to the increase in both sales of digital media as well as an individuals ability to illegally obtain and distribute this media. This paper will offer an alternative algorithm for performing audio file encryption that can also be applied to securely encrypt other media such as video, picture and text files.

Chapter 1: General Introduction

1.1 Encryption

There are different stages of the encryption process that a file must go through before it is fully encrypted. The *plain-text* is the data contained in the original unencrypted file. The plain-text does not refer specifically to text form instead it refers to the unaltered data available when the original file is opened. Some alternate file types are audio, video, or an image files. The plain-text is the binary representation of the digital media containing the sensitive information. The *cipher-text* is the result of applying the encryption algorithm to the original file. The *decryption key* is the set of values that when XORed with the cipher-text will return the original plain-text.

Definition: What is Cryptography?

Cryptography is "the science of developing cryptosystems that encipher and decipher data, among other things" [19]. Data is modified by means of applying a cipher to the plain-text in order to render it useless to unwanted parties. We encrypt messages with the intent of keeping unwanted individuals from accessing the information. Encryption is much more secure than the initial substitution of entire words, this is because when encrypting digital media we are able to break the message down into the bit level and encrypt each individual bit.

Definition: What is Encryption?

Encryption is the process modifying the contents of a file so that it will be unrecognizable and therefore unusable to an unauthorized party. This is achieved by applying a cryptographic function to the message. Different levels of encryption can be applied to the message depending on what the author of the message requires.

Areas of Use

Encryption played a significant role in times of war. Prior to the development of encryption a less secure method of message passing called steganography was employed in which messages were passed secretly by trying to hide the presence of a message. The Greeks often carved messages into the wooden writing tablets that they then covered with wax. The tablets were originally intended to be reused by writing in the wax covering, so they were able to pass messages unnoticed by hiding the message under the more common writing medium. Below is an example of an ancient wax writing tablet. It is easy to see how messages were written in plain text by scraping them into the wax covering. However when the message was carved into the surface of the wood, the wax provided an opaque covering hiding wooden surface of the tablet.

Figure 1-1 [14]

Another example is writing messages or code words on the jewellery worn by the message carrier. An additional method was carving messages into a soft piece of wood and then immediately pounding the soft wood with a hammer to temporarily erase the message. The message could then be sent undetected and would not be visible until the wood was soaked in water. These methods of message hiding were proving to be inadequate because they simply relied on unwanted parties not knowing the location or the method of extracting the secret information. If the location of the message was found such as the wax tablets or the jewellery then the message could easily be read. It was clear that a more secure method of message passing was

needed. If the location of a message was discovered but the message was encrypted then the message would remain unknown.

The necessary level of security provided by the encryption is dictated by the type of file as well as its contents. Files containing classified information require that encryption be applied to the entire file rendering it unrecognizable to all unwanted parties. Examples of such files would be image files containing maps, text files containing plans for a motion, audio files containing conversations or orders for a plan of action and also video file containing valuable location and layout information. However not all files contain classified information and the purpose of applying encryption is not to prevent the general public from ever knowing the contents of the file, but instead to limit the distribution of the decrypted file only to those individuals who have paid for that product. Examples of such files would be software, and digital media such as audio, video, image, and text files. The internet has created a way for the author of music, videos, literature, software, and art to distribute and sell their work without having to create hard copies on CDs or DVDs, involve a publisher, or create a physical print. Producers benefit by reducing the costs required to create a physical copy of the product as well as any necessary updates that may need to be distributed. However, this new mode of transferring media also allowed a new method of theft, it no longer

required the physical copies be stolen, and does not limit the number of copies distributed to the number of copies stolen. Once a copy of the original digital media has been obtained, thieves are able to create and distribute an unlimited number of copies from one original file.

One of the most common applications of encryption today is to commercial files. The intent is to mass distribute public information as quickly as possible to the greatest number possible. In order for this to be beneficial the information must be ensured a safe transfer from shipper to receiver or seller to buyer. Unlike before when encryption was used only in confidential correspondence where the cryptographer was limited to produce a product or service that was tailored to one very specific client, cryptographers now have a variety of clients with diverse needs. The new found demand for various levels of encryption to be applied to different types of media has allowed cryptographers to try new methods of encryption as well as increased the competition yielding better testing which in turn results in a higher standard of security.

The ease of online information interchange has created an increase in the need for information protection. This protection may be required by an individual user or an entire corporation. The single user may intend to share sensitive data only with a specific individual. In this case the individual could

purchase a program to encrypt the information for them. The corporations large or small may want to distribute information such as a product or service without concern that an unwanted party may steal and distribute duplicates of the information. The Windows operating system is an example of a software product, and Symantec's Norton Anti-Virus is an example of a software service that may be distributed either online or by a physical hard copy that could be illegally obtained and distributed. Smaller businesses as well as businesses who intend to distribute non-private information may prefer to purchase a program that will perform the encryption for them while larger companies or companies with very sensitive information may choose to develop their own. In the case of audio files most music production companies are more concerned with developing the music and marketing the artist than developing a method of encryption and would prefer to purchase a program with an existing encryption algorithm that meets current security standards.

1.2 Cryptanalysis

Cryptanalysis is "the study of breaking cryptosystems" [19]. A cryptographer develops methods of cryptography, while a *cryptanalyst* looks for ways to crack those methods. Most cryptanalysts are actually *cryptologists* meaning they study both ways to create encryption algorithms as well as

ways to break them. Cryptanalysts are often referred to as hackers or crackers since their purpose is to hack or crack the secret code. The best way to break an encryption is to understand how it works, so it is necessary to study methods of encryption instead of just developing algorithms that apply a brute force attack. Knowledge about an encryption algorithm is the key to cracking it in a minimal amount of time. Even a seemingly small clue may reduce the time required to discover the key exponentially. Having access to information such as the cipher text and the corresponding message text could prove to be very useful. If the same transformations are applied to the message text in every case then this one-to-one mapping of characters could be very helpful. When analyzing the cipher text many hackers look for repetition of characters. Most languages have specific letters that occur at a higher rate within that language than other letters. This distinction can aid hackers in developing the secret key. For instance if they are aware of the language the message being sent is written in hackers can look for the two or three most commonly used characters in the cipher-text and may be able to decrypt a portion of the message. Of course this method only applies to messages being passed in text format and would only work if the method of enciphering each letter remained constant for the entire message. An example of that would be a substitution cipher in which an instance of a letter would

always be replaced by only one corresponding letter. Cryptographers noticed that this method of hacking based on repetition of characters was increasingly popular so they began including dummy values in the cipher-text in an attempt to disable cracking the cipher by monitoring the repetition of characters. Another method used to confuse cryptanalysts was to map more than one cipher-text value to the same message-text character. This did decrease the ability of cryptanalysts to break the encryption based on inspection however if a copy of the message-text and the corresponding cipher-text were somehow made available to the cryptanalyst they may be able to derive the entire key if the message was long enough and contained a large number of different characters as well as repetitive characters. Below is an example of a simple substitution cipher with additional cipher-text characters for each plain-text character.

A - xyz	**G - fgh**	**M - nop**	**S - vwx**	**Y - def**
B - uvw	**H - cde**	**N - klm**	**T - stu**	**Z - abc**
C - rst	**I - zab**	**O - hij**	**U - pqr**	
D - opq	**J - wxy**	**P - efg**	**V - mno**	
E - lmn	**K - tuv**	**Q - bcd**	**W - jkl**	
F - ijk	**L - qrs**	**R - yza**	**X - ghi**	

For this cipher the capital letters are the plain text values and the cipher text values to the right of them are to be used for the first, second and third instances of the plain-text values. Once the fourth instance is reached the first value of the cipher text may be reused. Below is an example of an encrypted message using the given cipher.

plain-text :

MEET AT NOON BRING BACKUP WILL TAKE CITY AT DUSK

cipher-text:

nlms xt khil uyzmf vyrtpe jaqr tzun sbsd xt oqvv

possible decrypted cipher-text:

using only first instance possibilities:

mevt ak nofe brivg srckup wzlc kibm tqty ak dlss

using second instance possibilities on the partially decrypted text:

meet at noon brieg backup will trkv czcy at dusb

using third instance possibilities on the partially decrypted text:

meet at noon bring backup will take cicy at dusk

using fourth instance possibilities on the partially decrypted text:

meet at noon bring backup will take city at dusk

It may take more than four tries to decrypt this cipher-text, in this case I only retried the letters that I already knew were not part of the original

plain-text. The receiver would have no way of knowing which letters were the correct message-text characters.

This could cause minor problems for the deciphering party even if they possessed the key because they would not know what instance of the plain-text character created the corresponding cipher-text character. This is because the cipher-text characters could represent the first instance of one plain-text character or the second or third instance of a different plain text character. This is a small price to pay for the increased level of security it provided against unwelcome parties.

Another method used in the early stages of encryption was substituting only the vowels in words with either a one-to-one mapping of cipher-text character to each vowel or many-to-one as in the example before. This method seemed secure at the time due to the lack of knowledge of methods of encryption however in retrospect it reminds me of a simple game of hangman.

Those methods are outdated and digital media is now encrypted at the bit or sometimes byte level and occurs as a result of applying an algorithm. The algorithms no longer use static values to encrypt the messages but can instead use values created while the algorithm is running and the original message is being read immediately before encryption.

1.3 Attacks

Outlined below are a few of the many types of attacks used to break encryption algorithm. The methods below realistically describe information that a hacker may gain access to, while other methods which are not listed require the hacker to be given access to both cipher-text and corresponding plain-text. Circumstances in which a hacker gains access to both cipher-text plain-text and messages would occur if the algorithm had already been cracked and was later published or if someone intentionally leaked the information.

Brute-Force Attack

The brute-force attack refers to the method employed when little or no information about the encryption algorithm is known. Instead of approaching the decryption of the cipher-text with some known facts about the algorithm, it simply tries every possible case until the message is decrypted. In order for this to be useful a record would have to be kept of which method used to break the encryption was the correct method. This is not a very effective method because there are many ways the data could be encrypted. Encryption could be applied to segments of bits or bytes, and may not be applied to the entire file. If a hash function was applied the encrypted information may also not be in the order that the original message was. If the

encryption is applied more than once to the data then it will be impossible to know which sets of partially decrypted bits are the partially encrypted data so again the entire set of partially encrypted data would have to have the brute-force method applied to it as well. If different keys are used for encrypting the each segment of information it will require even more time to crack.

Below is an example of the number of possible ways a set of bits can be formed by applying the XOR operation to two sets of bits of the same length. If 011000 is the result of two binary numbers that have been XORed together then there are $2^n = 2^6$ possible ways to get that result.

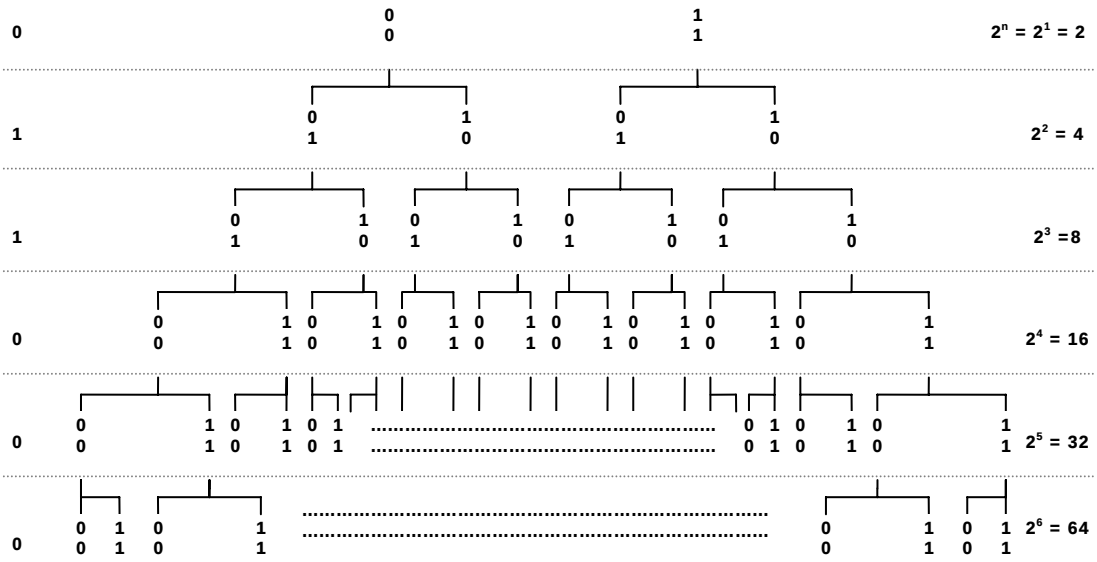


Figure 1-2

Cipher-text only Attack

In this attack the cryptanalyst has gained access to various cipher-texts and attempts to break the code by comparing similarities and differences between the cipher-texts.

Known-plaintext Attack

In this case the cryptanalyst has access to various plain-texts as well as the corresponding message-texts. The cryptanalyst is then able to deduce a key by comparing the many plain-texts and cipher-t

As mentioned before knowledge about a particular encryption algorithm will help minimize the time required to break the algorithm. Due to the recent increase in terrorist activity it has become important for the government to not only protect the information they wish to send and receive but also for them to be able to intercept messages being sent by terrorist groups. Many of these groups rely on the internet in order to reach many people over large distances while still retaining their anonymity. These transmissions may be intercepted but many of them are encrypted making it necessary for the government to employ cryptologists to decipher them.

Chapter 2: History of Encryption

Encryption as we know it today is the result of applying cryptography messages. Cryptographic substitutions of pictures and symbols for words or ideas have been used as since 2000 BC and possibly earlier. They were used by Egyptian scribes and are often seen on tombs, telling the story of the pharaoh who was buried there. Unlike encryption the purpose of this application of cryptography was not to cause the message to be unreadable, but to increase the interest and pay respect to the deceased.

Cryptography was employed as a form of secure message passing between two or more parties and is renowned for the significant role it played in times of war. Encrypting the messages was the only way to guarantee that confidential information could be sent and ensure it remained confidential even if it was intercepted by an unwanted party. Encryption was originally developed to be used in royal and military correspondence. The sensitive nature of the information being passed was partially protected by the lack of knowledge about the methods of encryption. The study of cryptographic methods has only recently been made available to the public due to the demand for stronger algorithms as well as the new areas of use.

Before encryption was developed, messages were hidden by disguising the message being sent. Invisible ink was used either to write an entire

message or to pass a secret message hidden in a seemingly harmless one. This can be achieved writing a message in plain text on the parchment and then placing dots with the invisible ink over characters in the plain text message. In either case a solution such as vinegar, lemon juice, sweat or saliva could be used and would be readable only once it was heated over flame. Below is an example of a letter written entirely in invisible-ink.

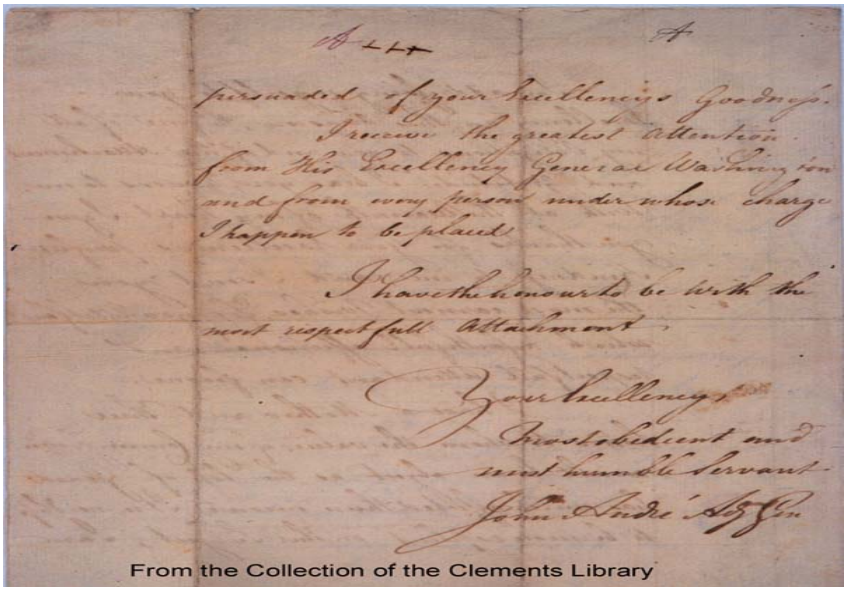


Figure 2-1 [9]

Alternative ways of sending messages undetected utilized the media in a different way then was originally intended. Messages were written on the earrings or jewellery worn by the message carriers would allow the message to remain undetected. Simply finding a creative way to hide the messages was considered sufficient then, but as more people became aware of different

forms of message hiding it was clear that a secure method of message passing was necessary.

Methods of encryption were being developed but very little was known about the process and many people referred to it as magic because the process of decryption was beyond their comprehension. Encryption was used mainly for royal and military correspondence, but was sometimes practiced by private parties who were conspiring against them. Individuals who studied ciphers and encryption independently were often subject to suspicion and considered to be a threat. This caused concern that these private parties may decrypt their royal correspondence and even publish the method they used. In many cases these individuals were harshly and publicly punished in an attempt to deter anyone else from practicing or considering practicing cryptography. Substitution ciphers were developed in which each letter of the alphabet had a corresponding cipher value that when applied to a message yielded an unrecognizable cipher-text. These were considered secure at the time but with the increase of individuals employing techniques of cryptography came an increased number of individuals dedicated to developing techniques to break these ciphers. Ciphers which were once thought to be unbreakable were being discovered and the messages they were employed to hide were exposed. It was clear that in order to create an

effective cipher it was necessary not only to try new methods of encryption but to perform cryptanalysis as well.

With the increased knowledge and application of encryption came the realization that in order to remain an effective military force it was necessary to study and employ methods of encryption. All over the world military leaders began to study and employ methods of encryption as a military tactic. Until this point messages were encrypted and decrypted entirely by human means. This method was both time consuming and not very secure.

Arthur Scherbius a German electrical engineer recognized the need for a more secure method of encrypting messages that required less time to encrypt and decrypt. In 1918 he patented a device which would later be modified and renamed the Enigma. The Enigma was an electro-mechanical device that could be used as a stationary device, powered by plugging it into a wall outlet or it could be used as a mobile device which was powered by a battery. The first version of the machine which was created by Scherbius was turned down by the German Navy due to the size and weight. Eight years later the German Navy did accept an adapted version of the Enigma which they called Funkschlüssel C. While many versions of the Enigma were developed and their method of encryption deciphered it played a major role in the way we approach encryption today. It allowed messages to be

encrypted as well as decrypted with increased speed and accuracy. Arthur Scherbius took the first step in implementing encryption by mechanical means. Below is a diagram of a simplified version of the Enigma machine.

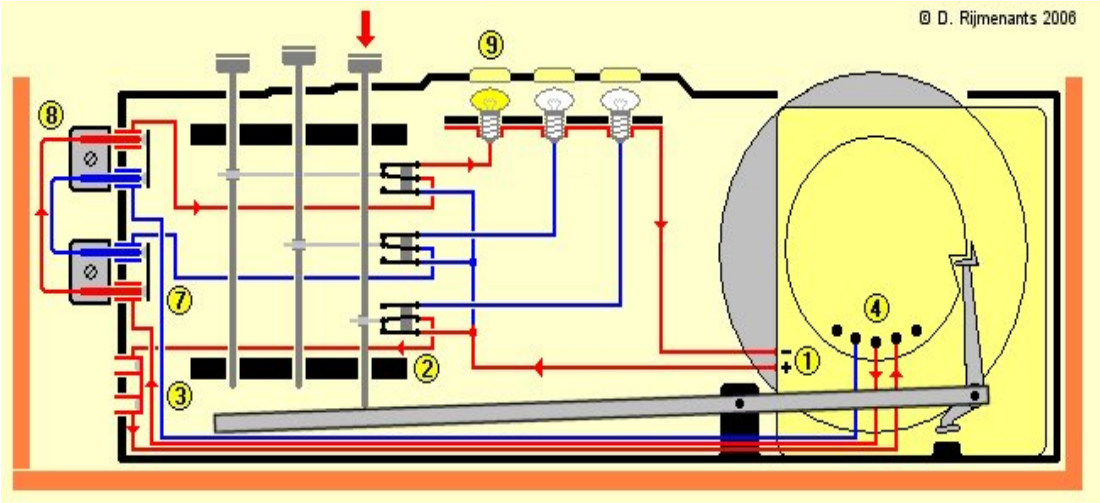


Figure 2-2 [11]

With the rise of technology and the development of digital media came a new more secure method of encryption. Encryption could be applied to binary characters that make up the symbols used in regular correspondence. Because each character was comprised of a series of one's and zero's an algorithm applied to the raw data would treat each character equally. In other words instead of mapping one alphanumeric character to another it could work on the data as a stream of identical characters. An algorithm could be applied that performed a transformation to this data and even change the sequence that the data arrived in. This helped to disable previous methods of

decryption which relied on studying the sequence and structure of the message in order to decrypt it.

This method of encryption is now applied to a broad range of information as well as various media. The content of this media ranges from highly sensitive government information to publicly known and distributed information such as art, literature, and music. Encryption is now applied to digital media in the previously mentioned areas in order to protect against digital theft. Different levels of encryption can be applied depending on what is required by the author of the file as well as the nature of the information contained in the file. Lower levels of encryption may only degrade the quality of the information contained in the file while still allowing it to be sampled or viewed. This level of encryption is often applied to text, audio, video and picture files that contain public information that the author wishes to sell. Allowing a portion of the media to be viewed or listened to gives consumers the chance to try the media without committing to buy it. High level encryption would be applied to corporate or government text, audio, video or picture files that contain sensitive information. High level encryption is also used to protect against identity theft on websites that offer financial services or deal with identification information such as social insurance numbers, point of sale terminals and bank machines.

Chapter 3: Existing Encryption Algorithms

Outlined below are a few of the most well known encryption algorithms that were developed and for a period of time were considered to be the standard.

RSA Encryption Algorithm

Developed by Ron Rivest, Adi Shamir, Leonard Adleman the name of the algorithm was derived from the first letters of their last names. It employs both public and private keys. Two large random primes p and q are generated and a modulus n such that n = pq. The private and public exponents are then calculated. To encrypt a message the sender must acquire the receiving party's public key. The sender creates the cipher text such that $c = m^e \bmod n$ and sends the corresponding cipher text c to the recipient. The recipient then applies the private decryption key to the cipher text such that $m = c^d \bmod n$.

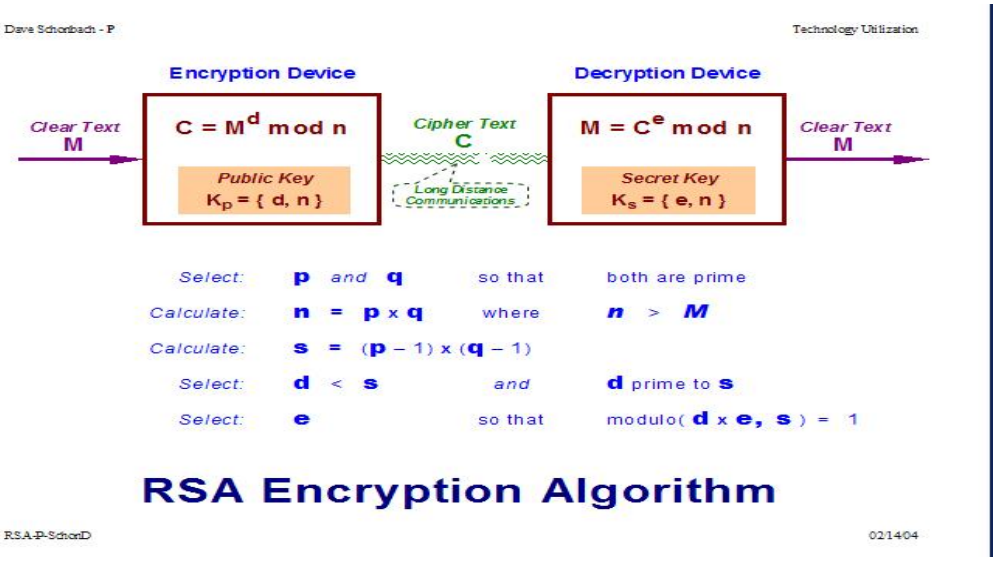


Figure 3-1 [12]

Example

1. Select two large prime numbers p and q:

p = 59 q = 71

2. Compute n = pq :

n = pq = (59*71) = 4189

3. Compute totient $\phi(n) = (p-1)(q-1)$:

$\phi(n) = (p-1)(q-1) = (58)(70) = 4060$

- Integers are said to be coprimes when they only share a factor of 1 or -1.

- The totient for a positive integer is the number of positive integers ≤ to that integer that are also coprimes with the integer.

4. Select an integer e where $1 < e < \phi(n)$ and e and $\phi(n)$ are coprimes:

e = 31 1 < 31 and 31 is not a factor of 4060

5. Compute d where $de \equiv 1 \pmod{\phi(n)}$:

d = 4191 31*4191 = 129921 = 1 + 32 * 4060

This generates the public key of n = 4189 and e = 31

$c = m^e \bmod n = m^{31} \bmod 4189$

and a private key of n = 4189 and d = 4191

$m = c^d \bmod n = c^{4191} \bmod 4189$

For a message m = 624 we encrypt by calculating:

$C = 624^{31} \bmod 4189 = 319$

To decrypt ciphertext c = 319 we calculate:

$m = 319^{4191} \bmod 4189 = 624$

SHA - 1 Secure Hash

The second version of five revisions, Secure Hash Algorithm-1 was released in 1995 is a hash function that converts a string of undetermined length to a string of a predefined length to create a message digest or a fingerprint of the original message. A collision is when a hash function produces the same output for two unique inputs. Collisions are an acceptable possibility for some hash functions but a high quality or strong hash functions should not have any collisions. SHA-1 produces a 160-bit digest for messages less than or equal to 2^{64}-1 bits. Collisions have been found in SHA-1 but the attack requires massive resources and is not realistic. SHA-1 has been improved upon by lengthening the digest, these versions are referred to as SHA-2.

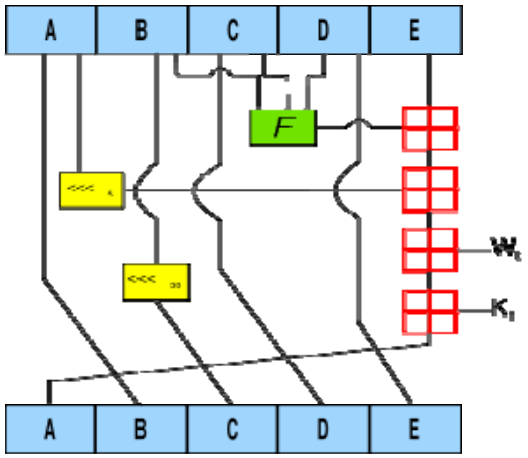
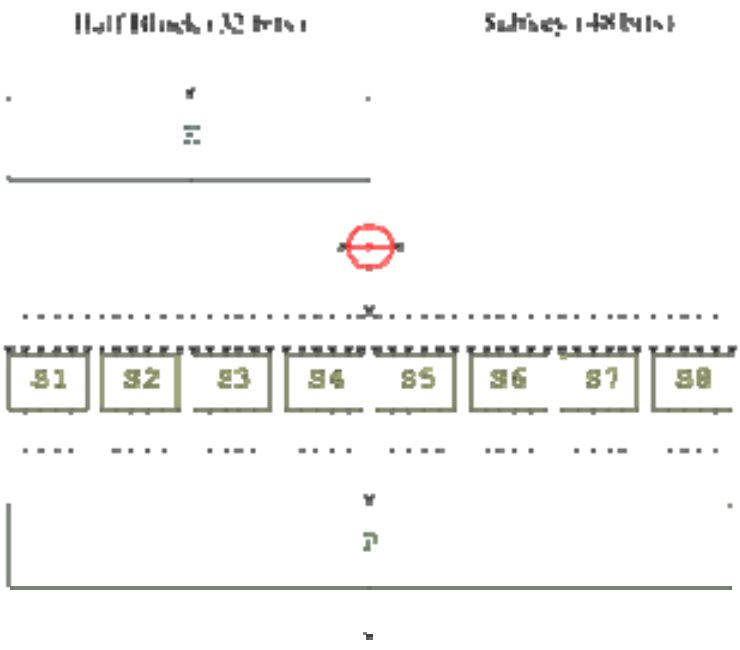


Figure 3-2 [18]

DES – Data Encryption Standard

In 1972 the National Bureau or Standards defined the need for a government wide standard of encryption. They began accepting proposals but did not accept any. Two years later they issued a second RFP and this time found a suitable Data Encryption Standard. Developed in 1973 by a team of twelve IBM employees, based on a team member Horst Feistel's algorithm it was to be the new standard. The DES was a block cipher which separated the message into 64-bit blocks and employed a 56-bit key. Due to the short key size DES has been broken using the brute force attack and has since been redefined as the Triple DES which applies the encryption three times and finally the AES or Advanced Encryption Standard.



The Feistel function (F function) of DES

Figure 3-3 [17]

Types of Encryption

Symmetric

Symmetric encryption also known as private-key encryption uses the same key to encrypt and decrypt the information. It is viewed as less secure than public-key encryption because if the encryption key is discovered the unintended party will also posses the decryption key. However it is a lot less computationally intensive so it requires less time and resources to implement.

Asymmetric

Asymmetric encryption also referred to as public-key encryption employs the use of both a public and private key. Asymmetric encryption is considered more secure because it is computationally infeasible to factor the large primes. A worked example of the RSA implementation is shown above.

Chapter 4: My Approach to Audio File Encryption

4.1 General Discription

"*keyboard drivers:* A term for pirated software and such. Generally used when trying to disguise an attempt to find pirated software etc.

> Does anybody know where to find some good keyboard drivers?
>
> That site has some good keyboard drivers. You should check it out." [15]

Keyboard drivers a single entry in the Urban Dictionary defines a now commonly used term in chat rooms world wide. This term which refers to *pirated* or illegally obtained media of various types enables the individual to search the internet for pirated media without alerting security programs. This definition makes it clear that in order to ensure our files do not become pirated software we must continue to develop more secure methods of encryption.

The approach I took to encryption is a result of the increase in pirated software. It is easy to see the growing demand for protection against the actions of individuals who choose to use the resources for negative purposes. Because of the rising number of users who are illegally sharing software and the security measures employed to stop them, users have had to find alternative ways to locate and access the pirated media. These files do not contain government or military information and for that reason do not require

the same level of security. Instead the information contained in the file may be fully available to the public, but if an individual wishes to have a copy of the file to use as they wish they must purchase a legitimate copy. The level of security necessary for these types of files is determined by the file content as well as the owner of the file.

My approach employs an algorithm that uses a private key. Similar to the ECB (Electronic Codebook) cipher after separating the message into blocks it encrypts each block independently; ensuring that if one section of the message is deciphered that it will not affect the rest of the encrypted information. Unlike many ciphers which generate the values in the initialization block using the random function the values in the initialization block are the result of XORing together values from the original message. My program reads in the file and places each ASCII character into an index of the array. Each ASCII character is represented by two Arabic numerals displayed in Decimal, Hexidecimal, or Octal. Each Arabic numeral is represented in turn by four bits.

ASCII - RIFF

Dec – 82 73 70 70

Bin – 1000 0010 0111 0011 0111 0000 0111 0000

Each 512-character block is represented by 4096 bits is encrypted by XORing it with an initialization block of the same length. The initialization block is created by XORing hashed values from the original message. A unique initialization block is created for each 512-character block of the message. If one initialization block is discovered the other encrypted blocks will remain encrypted. This requires hackers to use the brute force approach on each individual encrypted section.

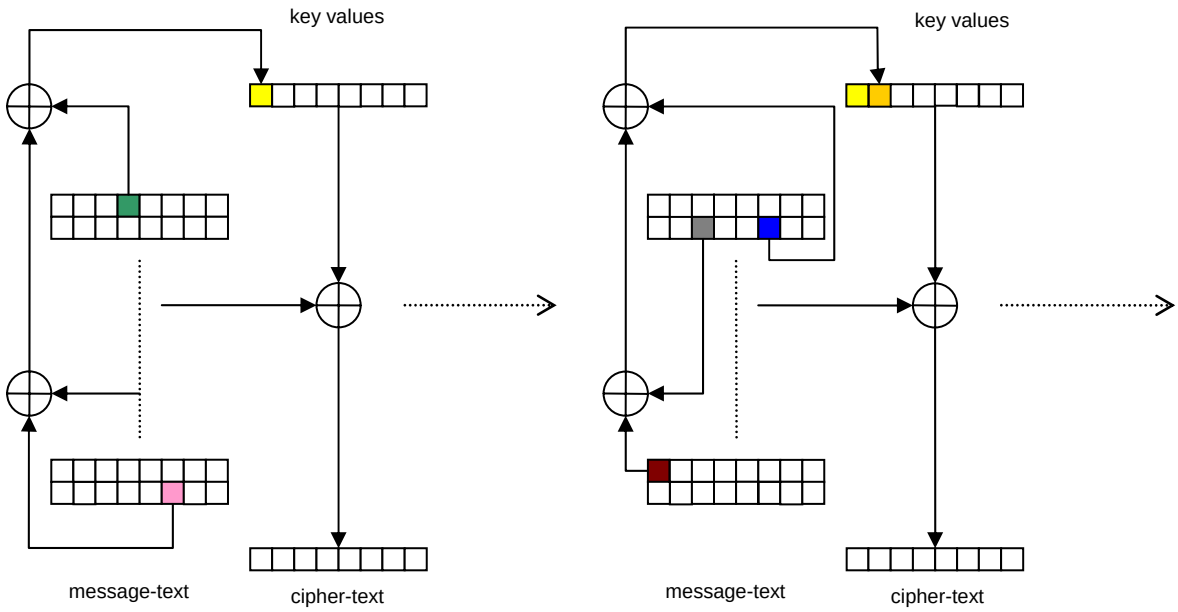


Figure 4-1

My approach to audio file encrypted was geared towards audio files which are intended for public distribution but at a previously defined cost to the user. This determined the level of security necessary as well as the manner in which the algorithm was applied. If the files had contained highly

confidential information it may have been advantageous to apply the encryption to the file multiple times as well as applying it to the entire data portion or even the entire file including the file header as opposed to only a part of the data portion. Instead because the purpose is to ensure a profit for the artist and production company it is even more beneficial if the file is only partially encrypted allowing the public to sample the music but without giving them access to the complete version.

4.2 Structure of a .wav File

The structure of a .wav file determines the format of the resulting file after encryption. Wave files have a file header that contains information about the file type and size of the file as well as the format of the sound portion in the data chunk. The header information is read by the program such as Windows Media Player or Winamp that intends on opening the file. If the correct information identifying the file as a .wav file is located then the program is able to open the file to be played. However, if the correct identification information is not found, then the program sends an error message to the user explaining the file is not of the intended format. When applying encryption to audio files if the entire file is encrypted the header information will also be modified. This means the header information identifying it as an audio file will no longer be readable to the program which

is trying to open it. If the intent of the party encrypting the file is only to disable the content from all unauthorized parties then this method is acceptable. On the other hand, if the party encrypting the file only wants to ensure that the full version of the file is not available to the public but does not require that the content be completely disabled then another approach may be considered. Most music files as well as video files have been created for public listening or viewing but are not intended to be distributed free of cost to the public. While these files contain no sensitive information they still require some form of protection against unintended distribution. In this case because the files were created with the intention that the files would be purchased it is more industrious for the files to be only partially encrypted. This allows the public to obtain copies in which only a portion of the file is able to be viewed or listened to. This marketing technique is evident in the way we now encrypt music files. In order to perform this type of encryption it is necessary that the portion of the file encrypted does not contain any of the header information. Audio files can be traversed using Multi Media Input/Output functions defined and provided by Microsoft. They provide a way for us to locate specific sections of the file which aid in determining entire file size as well as the size of individual sections within the file. The encryption can then be applied to the data portion of the file ensuring that

upon opening the file it will be identified as an audio file. A section of any length or multiple sections can be encrypted as long as they are not a part of the header portion of the file. The encrypted portion of the file may sound or appear differently depending on the program in which it was opened. Possibilities for audio files include but are not limited to a loud repetitive screeching, a short quiet static sound, a lack of sound and an abbreviated time frame for the encrypted section.

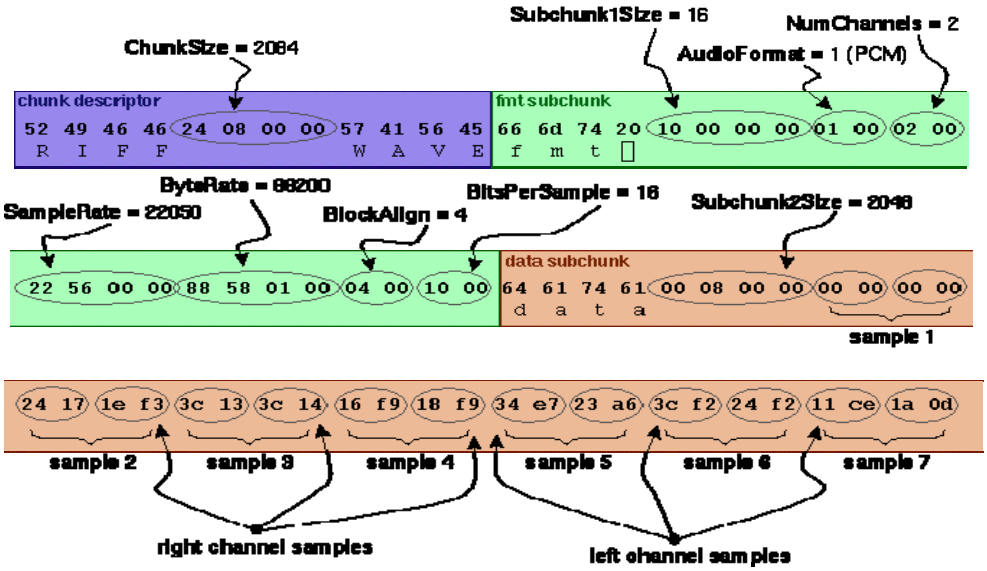


Figure 4.2-1 [16]

4.3 My Algorithm

My first version of the algorithm was tailored specifically to handle .wav files. It was important for the file header to be untouched so the file could be opened using common software such as Windows Media Player or Winamp. I used MMIO functions which are predefined and offered by Microsoft.

Information on them can be found at the MSDN website. They enable traversal of file sections and return information about the size of each section. That algorithm successfully encrypted and decrypted .wav files but was limited to only that file type. Once I had completed that version I realized that the only condition that had to be met to ensure the .wav file could still be opened and recognized by common audio file editor software was to leave the header file untouched. I wanted to see if the same encryption could be applied to other file types so I modified the algorithm and ended up with the last version which is given in the Appendix.

Traversing the Header File and Placing it in Segments

In order to enable this encryption to be applied to multiple file types I removed the MMIO functions applied the encryption to a portion of the file located after the file header. The file is encrypted by placing the file information into a two-dimensional array of fixed size. The dimensions of the array I used are segment [i][m] = segment [480][2048] which will accommodate a file with playing time of about four minutes which is the average length of a song. When I refer to a segment of the array segment[i][m] I am referring to the i value. The length of the file is needed in order to determine the number of array segments necessary to hold the values of the

original message. If the message is not length%2048 = 0 the remaining portion is padded with zeros.

Key Creation

The key is created while the message values are being placed in the allotted segments of the array. The key values are placed in the segments immediately following the last segment designated to hold the message. The key is created using values from the original message. This process begins at index segment[messagelength-keylength][0] and ends at index segment[messagelength][2048] and the values are selected only from segments that have already been populated with plain-text values. This is necessary to avoid using null values because the key is being created while the message is still being placed into the array.

Length of File

The length of file that can be encrypted is limited to the size of the array. The sum of the message length plus the key length must be within the size limits offered by the array. The values of the key selected are dependent of the current index of the array. This provides more security than selecting a fixed value and increases the difficulty in determining the locations of the values used to create the key.

Creation of the key and encryption of the file is done by multifilenecryption.cpp. In the case of a company wanting to encrypt many files they would use multifileencryption.cpp to encrypt the files and the intended receiving party would need a copy of multifiledecryption.cpp as well as the key to decrypt the file. The key created for each file distributed can be modified by the encrypting party prior to distribution.

Decryption

Decryption of the file is handled by multifiledecryption.cpp. In order to decrypt the file the decrypting party must have a copy of the encrypted file, the program to decrypt the file as well as the key.

Level of Security

In the section covering the brute force attack there is a diagram showing the number of possible ways to create an n-digit binary number by XORing two n-digit binary numbers. There are 2^n different combinations of same length n-digit binary numbers that when XORed together will result in a particular n-digit binary number. We can use this knowledge to develop a best-case and worst-case for cracking the encryption the encryption of a given section. This will give us an idea of how secure the algorithm is.

The cipher-text is the result of XORing the message with the initialization block. For this example let M be the message text, IB be the unique

initialization block and CT be the cipher-text. Since CT is the result of XORing M and IB we can get M by XORing IB and CT. The brute force method tries all possible combinations until it finds a successful combination. Assuming that the brute force algorithm starts with one extreme such as all 0's and if it does not find a successful combination eventually ends in a combination of all 1's. There is a chance that the hacker may have started with the correct combination and indeed have found the IB, but the probability of that happening would be $1 : 1.34 \times 10^{154}$ or $1 : 2^{512}$ if we had used 512-bit segments but we used 512-character blocks or 4096-bit segments. This decreases the probability of selecting the correct combination on the first try to $1 : 2^{4096}$. In this case they would still have to verify that it is the correct combination by XORing it with the corresponding section of the CT. If the encryption is applied n-times to the same section then the best case to crack it would be n-tries. This is nearly impossible to generate the correct combination of values with no previous knowledge of the algorithm or the message text but cannot be ruled out. A unique 4096-bit key is created for each segment to be encrypted. I decided to encrypt forty segments to this process would have to be repeated for each encrypted segment. The number of tries required to arrive at the correct combination must then be added to the number of tries required to arrive at the correct combination for each section. This means even

in the case that the correct combinations were chosen for each section then the best case would be the number of segments. The best case to crack the encryption of a file with n encrypted segments is n. The worst case for a file with only one encrypted section of 4096-bits would be 2^{4096} tries. The worst case to find the correct combination of 512-character segment for a file with m-segments would be $(2^{4096})^m$ tries. Each combination would still have to be checked by XORing it with the message-digest if it were a text file it could be opened and if the characters formed a readable message in a given language then the hacker would conclude the correct digest was found. However because we are dealing with audio files, if the entire file was encrypted including the file header, after XORing the possible array of initial values with the message-digest if the file was able to be opened as an audio file it was be fairly easy to determine that the correct array of values was recreated. In the case that the encrypting party encrypted only the data portion, the hacker would be required to open each file and listen to determine if the current set of values were in fact the correct set of initial values. The process would be incredibly time consuming for the system to create each file and for the hacker to listen to each individual file.

4.4 Example .wav

I used Cool Edit Pro Version 2.1 to visually analyse the audio data. If the file you are opening is an audio file but only a segment therefore it does not have the appropriate header section to identify the file type Cool Edit creates the header based on information that supplied by the user.

It allows you to specify the sample rate, number of channels and resolution of the file you are opening. Below is the waveform view of the audio file Wouldn't it be Nice by the Beetles which for ease of use I renamed 1.wav.

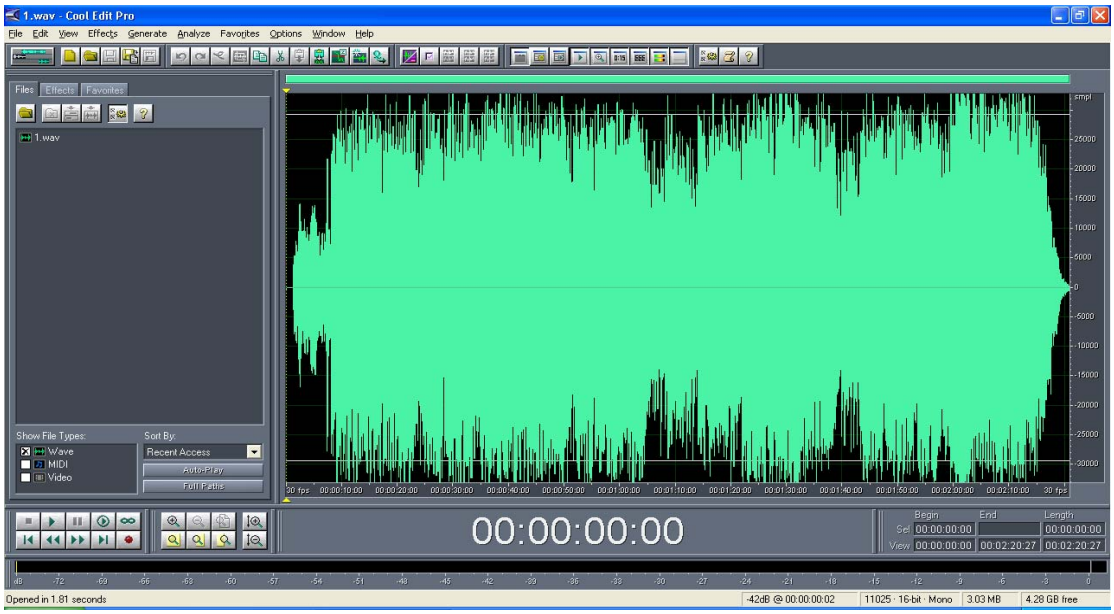


Figure 4.3-1

The image shows the file in its original unaltered form. The file is read and the values are placed in a multidimensional array. I output the contents of the

array that contained the file data to a file called message.wav shown below to ensure the file was being read accurately.

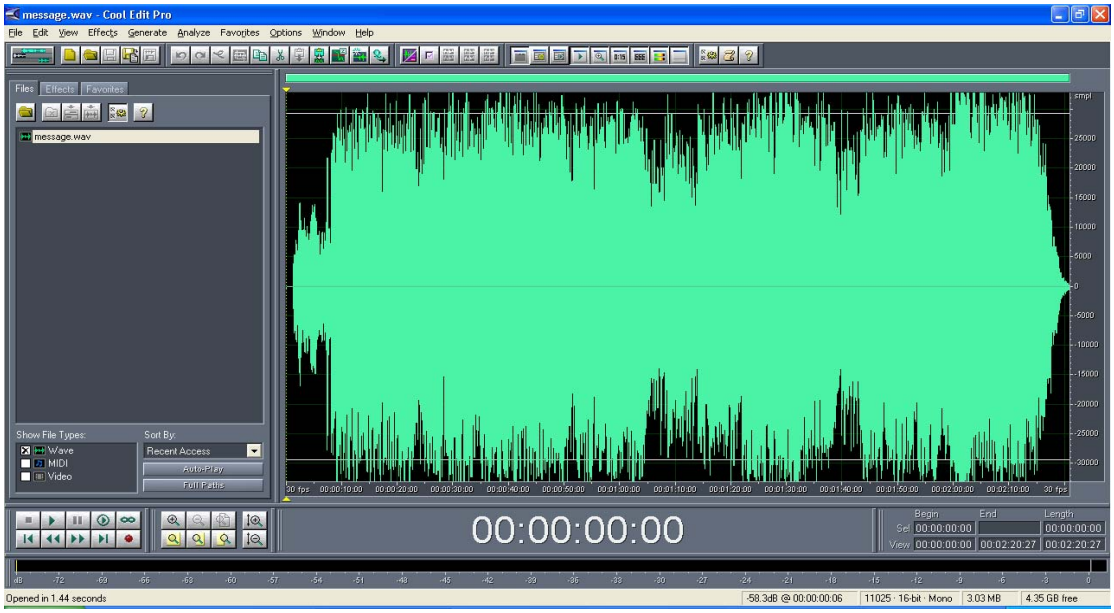


Figure 4.3-2

In order to explain and compare the steps of the encryption process I output various sections of the array to .wav files. These files are not all needed in order to encrypt or decrypt the file. The only files that needed are the original file which will be encrypted, the encrypted file encryptmessage.wav, and the key key.wav. The next file to be created is key.wav. Each key value in the array of the key.wav file is created by XORing two values from different segments of the file and then XORing the result with a final value selected from another segment of the file. This file XORed with a portion of the message will create the cipher-text or encrypted portion of the file.

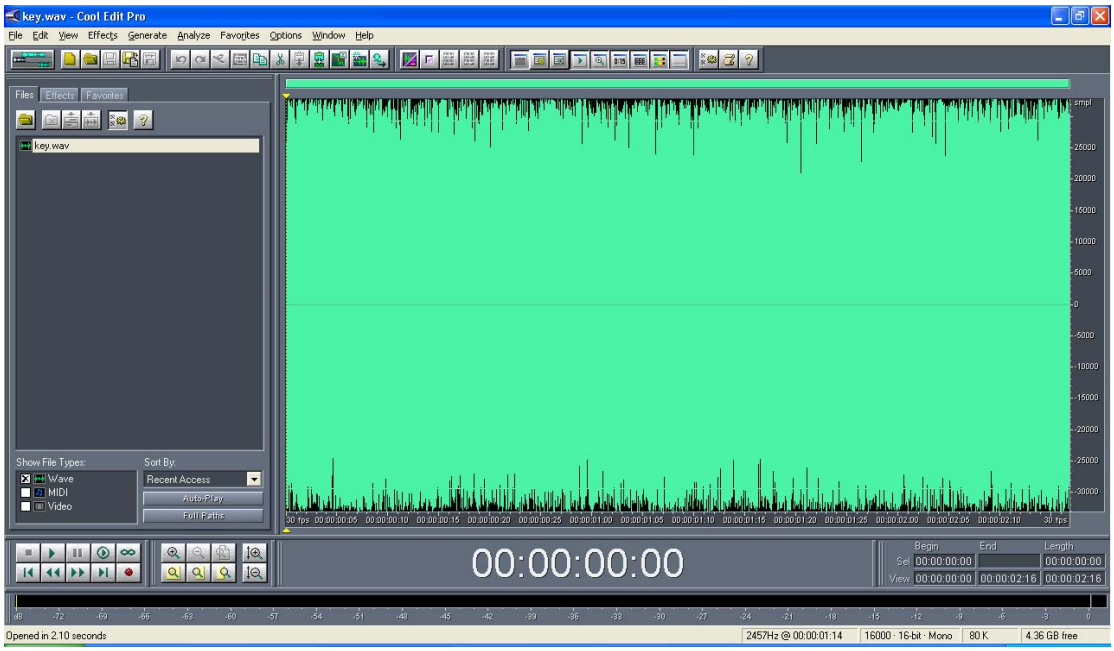


Figure 4.3-3

File key2.wav is simply to ensure that the values being read from key.wav are in fact the correct values. This may seem redundant but the decryption process is a direct result of the key values being XORed with the corresponding encrypted message values, using the bit before or after the actual key values will result in encrypting the file again but using an unintended key.

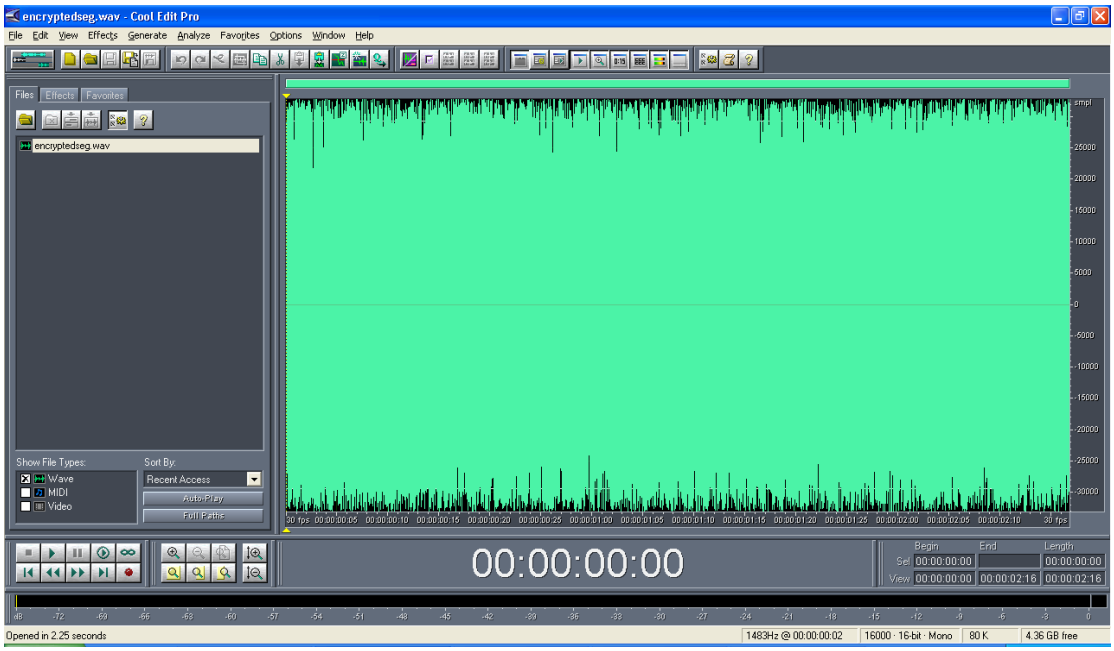


Figure 4.3-4

I created the file encryptedseg.wav to compare to key.wav to guarantee that the file was in fact XORing the values of the message and the key together instead of just replacing the message values with the key values. This file is created by narrowing the output of the file to only the segments that were encrypted. When the encrypted file is viewed in Cool Edit Pro the encrypted segments appear as a lack of sound but when I narrow the output from multifileencryption.cpp to include only the encrypted segments which are from segment forty to segment eighty Cool Edit shows the above file. As mentioned before each file will be recognized differently by different software. Another thing to note is even the length of the file or the time it takes to listen to it is modified by the audio file editor. When opening the encrypted file in WMP or Winamp the encrypted section is shortened and

sounds like a skip but in Cool Edit it shows a missing section when the entire file is viewed. If the zoom function is used in Cool Edit it still shows a lack of sound, it is only when the segments output to the file are narrowed in multifileencryption.cpp to show only the encrypted segments that we see and are able to hear the encrypted portion. CEP shows the length of the file in the minutes:seconds:1/30 of a second format.

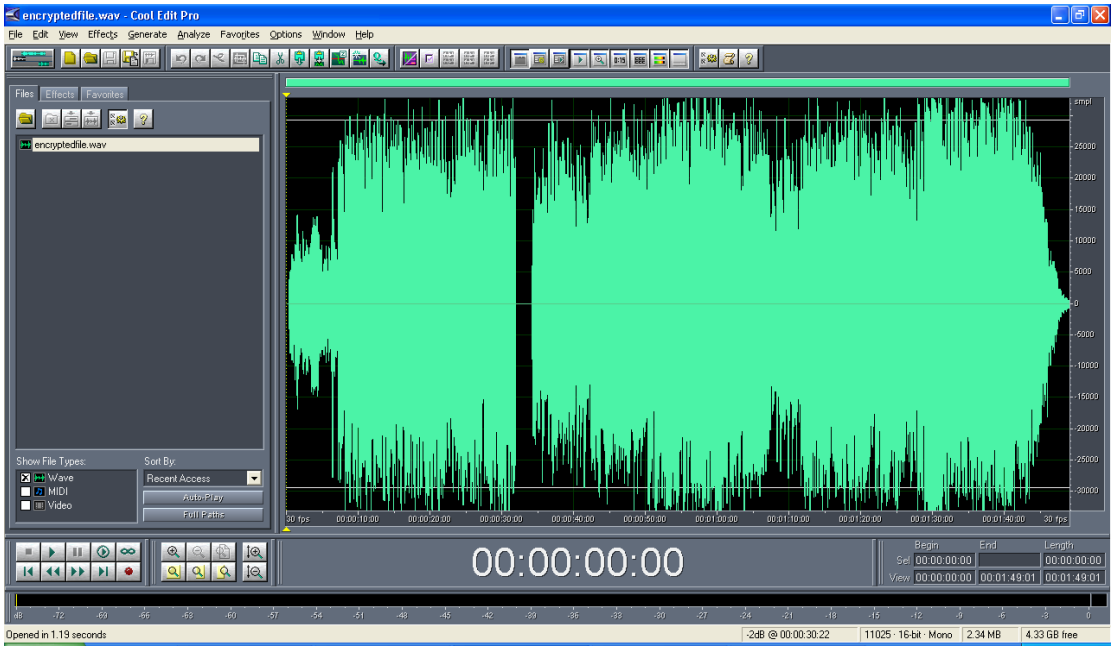


Figure 4.3-5

The original file length shown in the bottom right corner of the CEP screen is 02:20:27 while the length of the encrypted file is 1:49:01 this is a difference of 00:31:26 seconds. This means the key, encrypted segments, and decrypted segments should be the same length.

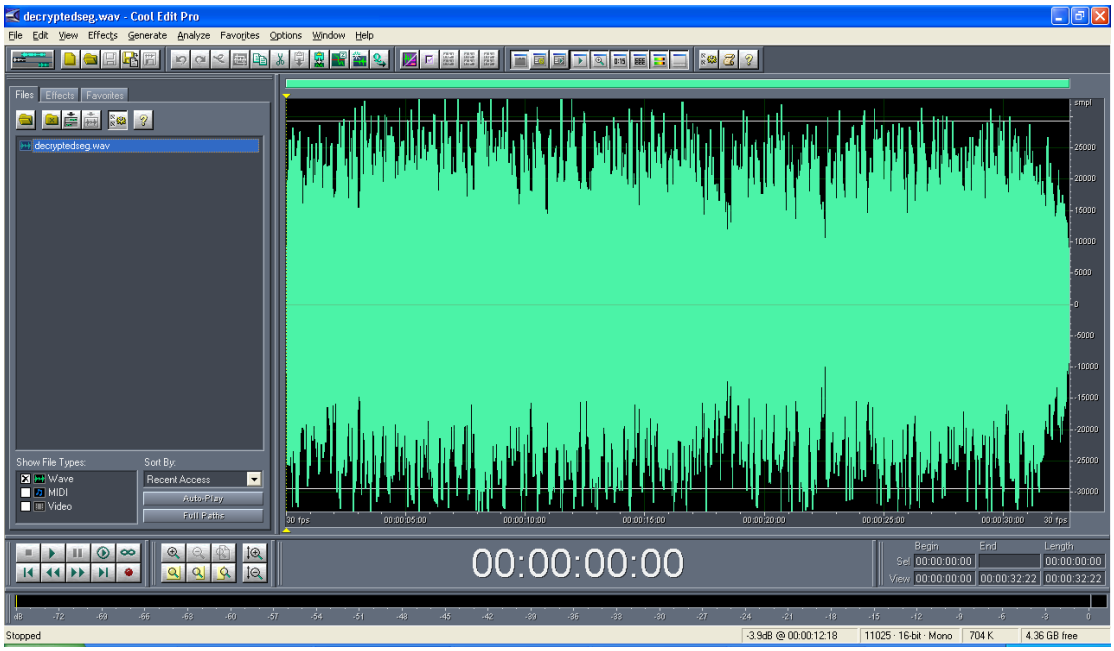


Figure 4.3-6

Instead even CEP concatenates the encrypted portion partially the decrypted segments are actually a length of 00:32:22 seconds and the key and encrypted segments are showing a length of 00:02:16. One 4096-bit segment is about 0.81825 of a second which should be fairly noticeable but because CEP shortens the playing time of the encrypted segments it is hardly noticeable. So even omitting one segment could go unnoticed in an audio file but must be checked using various media. Comparing file size was also a helpful method as well as the zoom options offered by Cool Edit that allow you to stretch out the wave file.

Below are images of the decrypted file in which I have selected an area to be examined. When a segment was missing the wave was displayed as a straight

line in the middle of the screen. This was very helpful however is only shown when the wave is magnified excessively.

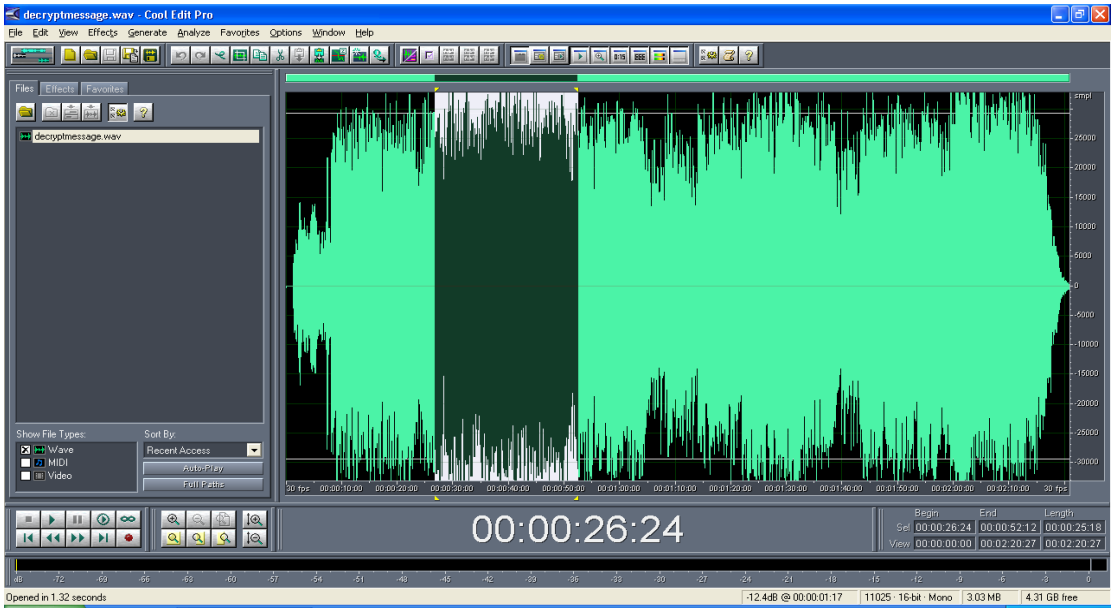


Figure 4.3-7

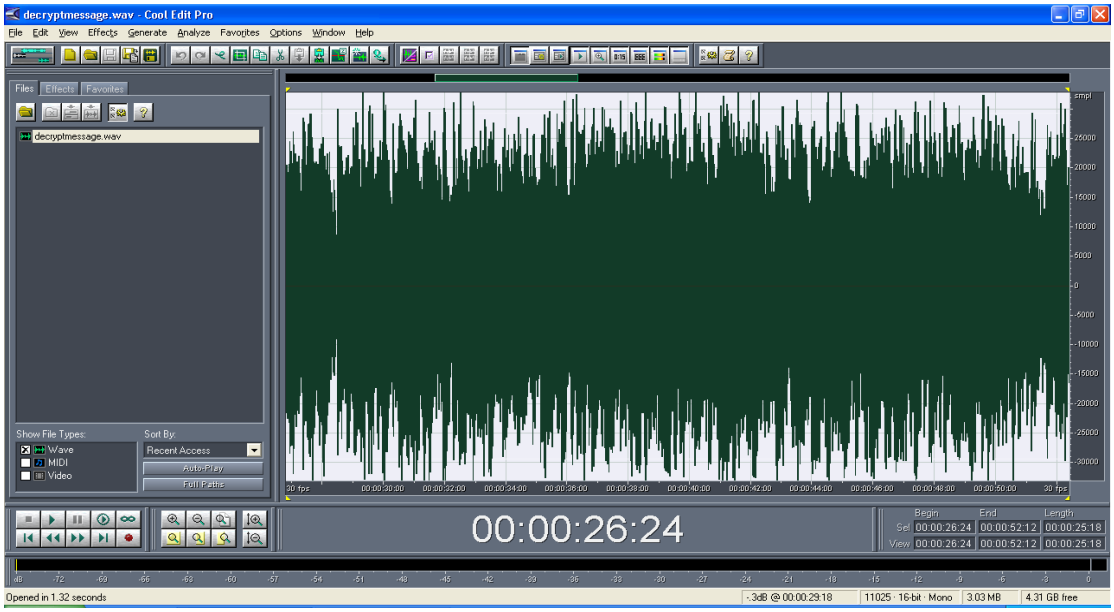


Figure 4.3-8

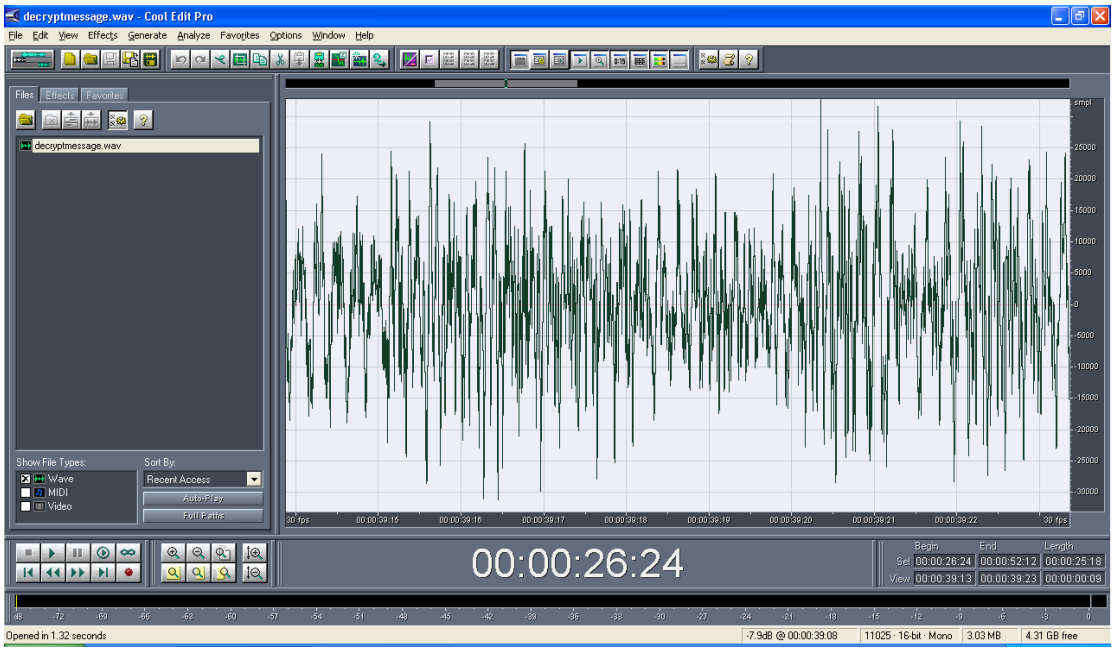


Figure 4.3-9

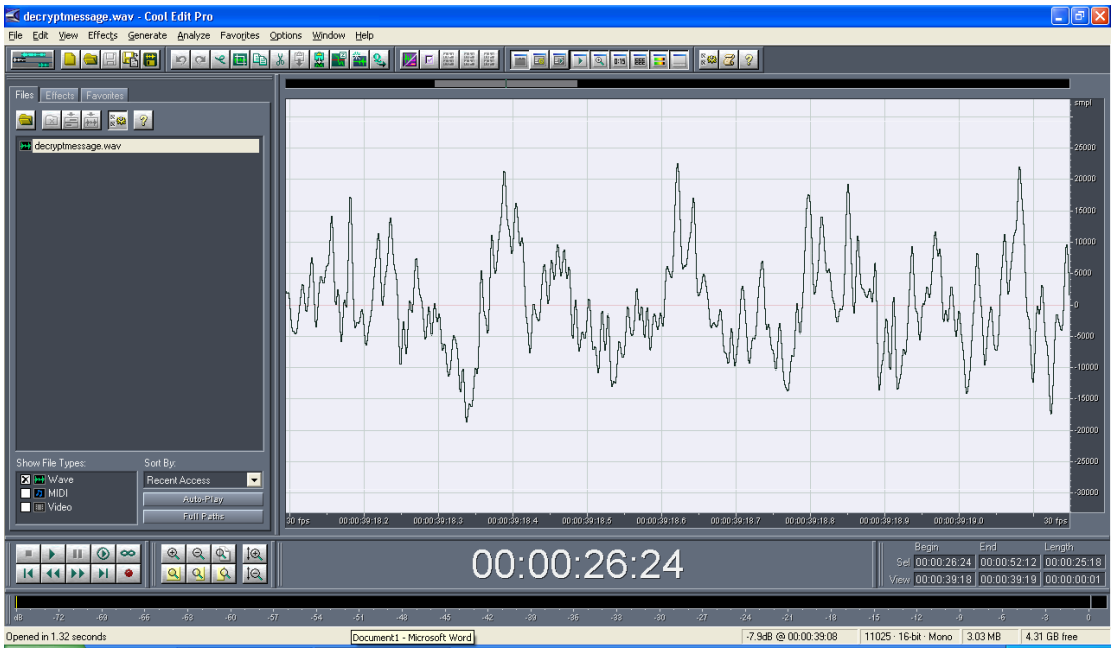


Figure 4.3-10

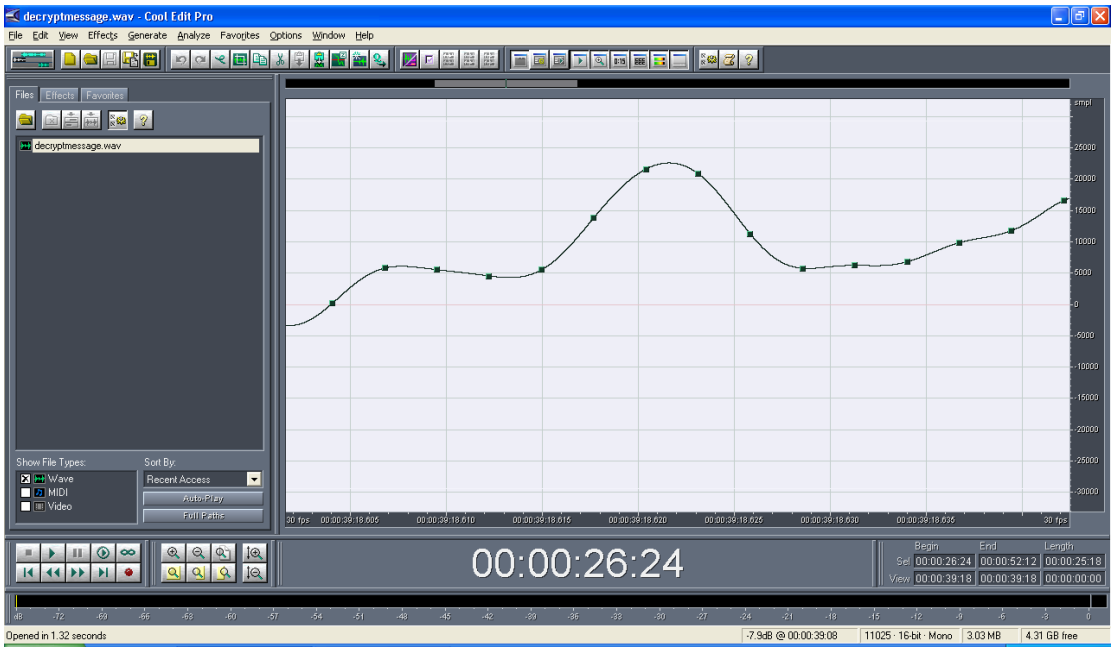


Figure 4.3-11

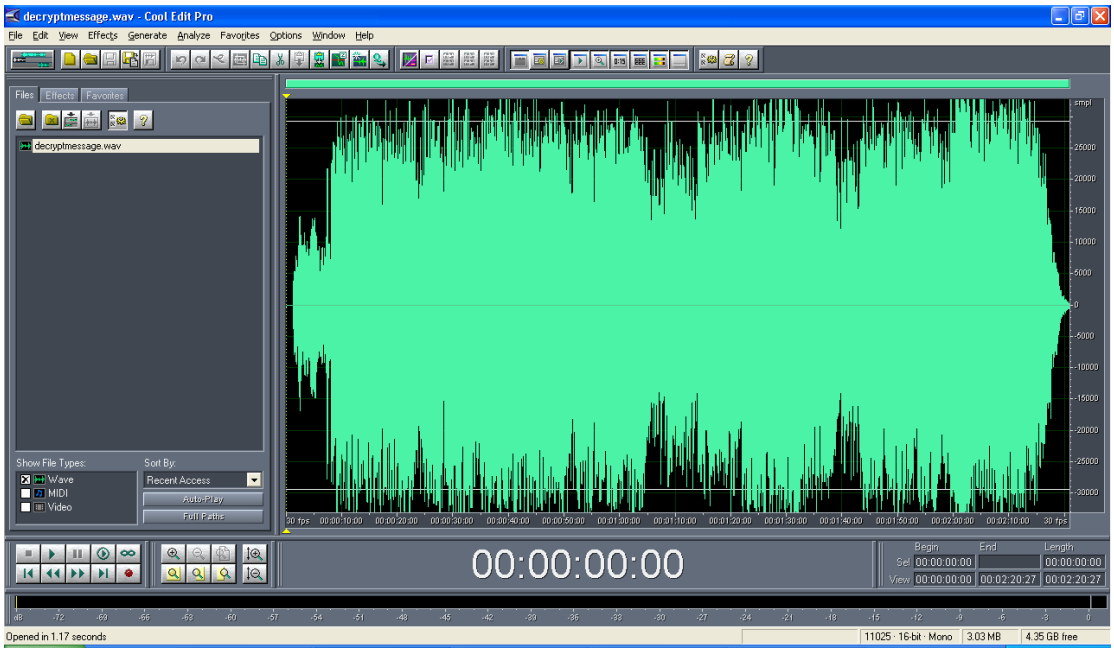


Figure 4.3-12

The decrypted file shows the same length as the original and can be verified by using a combination of the zoom functions, shown in the bottom left corner by the play

controls, offered by Cool Edit Pro. Below is an example of the results of applying this method of encryption and decryption to a .jpg file.

4.5 Example .jpg

ratchet.jpg

Figure 4.5-1

encrypted.jpg

Figure 4.5-2

decrypted.jpg

Figure 4.5-3

It is easy to notice differences when performing a visual comparison of .jpg file types.

Chapter 5: Future of Audio File Encryption

Future Uses

As the rate of digital media exchange increases, issues of privacy and identity theft will also increase and encryption will play an important role in ensuring digital media is exchanged in a secure manner.

Apple has introduced a new file sharing technique between Ipods that are in close range of each other. This means that users are able to sample music that other Ipod users have purchased and are currently listening to. The person sampling is allowed to listen to the full version of the track once after which the file is automatically encrypted and they must purchase their own legitimate copy if they wish to hear it again. This concept of previewing the file before purchasing is recognized in marketing which is why many songs and movies are only partially encrypted allowing them to view partial content but requiring them to purchase the rest. This technique of having full access to a product or service is employed in all areas of marketing from software to cell phone service and is now being used in methods of encryption.

The way we use encryption in the future and the areas we use it in are dictated only by the areas we use digital media and the ways we can apply encryption. Encryption is being applied in more diverse areas including

various mobile devices. As electronic development and capability continues to grow so does the need for encryption in these areas.

Chapter 6: Conclusions

In the example section for .wav files, when comparing the wave form data there were specific differences between the length of the encrypted segments, key, and encrypted file from the original file. This is an example of how important it is to use various media to ensure the correct data is being accessed. If I only used an audio editor such as WMP or Winamp I would not be able to see if the file was in fact being encrypted as opposed to just removing a segment. Another method is opening the files in notepad. This can be time consuming though based on the size of the file. This is another reason why I felt applying this encryption to files such as images would be important as well as helpful. When applied to images if the key is fairly simple such as only one set of XORed values the portions of the image may still be recognizable. It required comparisons between the wave form view of the audio file as well as .txt values of the .wav file and finally comparisons with .jpg to ensure that the algorithm was correct. I was actually sure it was correct until I applied it to a .jpg and the difference in the image was much easier to see than the difference in the wave form view of any audio files I encrypted.

Strengths and Weaknesses

The algorithm I designed has an increased key size and offers a unique key for each section being encrypted. The process of creating the key is not given in the decryption process so the user will not be able to derive the key if they only have the decryption software. If the hacker has access to a copy of the key, with 2^{4096} possible ways to get the combination of the key from two same length binary numbers, even if they found the correct values that created the key they would have no way of knowing which location in the file the bits were taken from. This means even if the key is leaked since each individual key value is created by XORing two 8-bit combinations of 1's and 0's it is nearly impossible to know which two characters yielded that key values. After finding even those two values because the key values are created by XORing two 8-bit numbers the result of that operation is then XORed again with another 8-bit value. Again even if the hacker did manage to select the correct 8-bit value they would have no way of knowing the location in the file that it came from.

((11010111 ^ 11101001) ^ 01101011)

= (00111110) ^ 01101011

= 01101011

Because the algorithm I created employs symmetric encryption the key is needed in order for the intended receiving party to decrypt the message. Unlike asymmetric encryption where the message is encrypted with a public key and the private key is only known by the receiving party, in symmetric encryption the key is created by the encrypting party and a copy is sent to the decrypting party. The method in which the key is delivered increases the potential of the key being leaked to an unintended party. Even though they may not know the method of encryption so they will not be able to decrypt other messages that have keys created by the same algorithm, they will be able to decrypt the message in which they have a copy of the key.

Having created an alternate method of audio file encryption using symmetric encryption and then modifying it so it could be applied to other file types I am still curious about changing the method of encryption to asymmetric encryption. Due to time constraints I am not able to provide both methods but it is an area that I would like to explore as well as what is required to break the symmetric encryption algorithm I created.

Bibliography

[1] "About the Interchange File Format (IFF)." Author uknown.
<http://www.borg.com/~jglatt/tech/mmio.htm>.

[2] "WAVE File Format." Author uknown.
<http://www.borg.com/~jglatt/tech/wave.htm>.

[3] Ball, Rouse W. W. "Mathematical Recreations & Essays." New York: The Macmillan Company, 1960.

[4] "Write operations in C++ Binary File I/O." Codersource 5 Jan. 2004
< http://www.codersource.net/cpp_file_io_binary.html>.

[5] D'Agapeeff, Alexander. "Codes and Ciphers." Oxford University Press 1949.

[6] "Reading/Writing .wav Files." DePiero, Dr. Fred. 8 Oct. 2002
< http://people.msoe.edu/~taylor/examples/wav.htm>.

[7] Hacking, Ian. "Logic of Statistical Inference." Great Britain: University Printing House, 1965.

[8] "Ratchet and Clank." Insomniac Games. 2003
< http://www.us.playstation.com/Content/OGS/SCUS-97268/Site/>

[9] "Spies of the Revolution." McGinn, Shaun. Vinci, Allison.
< http://edweb.sdsu.edu/wip/examples/spies/letter.jpg >.

[10] Mollin, Richard A. "An Introduction to Cryptography." Chapman & Hall/CRC, 2001.

[11] "Technical Details of Enigma." Rijmenant, Dirk, Cipher Machines and Cryptography. 2004 <http://users.telenet.be/d.rijmenants/en/enigmatech.htm>.

[12] "RSA Algorithm." Schonbach, Dr. Dave. 2 Sept. 2004 < http://www.schonbach.com/csw/051_classes/Word_Web/ RSA_Algorithm.jpg >.

[13] "Cool Edit Pro 2.1." Syntrillium Software. 9 Apr. 2003 <http://www.softpedia.com/get/Multimedia/Audio/Audio-Editors-Recorders/Cool-Edit-Pro.shtml>

[14] "Learning About Papyrology : Ancient Writing Materials : Wax Tablets." University of Michigan, Papyrus Collection. 2004 <http://www.lib.umich.edu/pap/k12/materials/wax.html>.

[15] "keyboard drivers." Urban Dictionary. 20 Apr. 2004 < http://www.urbandictionary.com/define.php?term=pirate+ software&page=2 >.

[16] "WAVE PCM soundfile format." Wilson, Scott. 20 Jan. 2003 < http://ccrma.stanford.edu/courses/422/projects/WaveFormat/ >.

[17] "Data Encryption Standard." Wikipedia, The Free Encyclopedia.

< http://en.wikipedia.org/wiki/Data_Encryption_Standard >.

< http://en.wikipedia.org/w/index.php?title=SHA_hash_functions &oldid=114064997 >.

[18] "Secure Hash hash functions." Wikipedia, The Free Encyclopedia.

< http://en.wikipedia.org/wiki/Data_Encryption_Standard >.

< http://en.wikipedia.org/w/index.php?title=SHA_hash_functions &oldid=114064997 >.

[19] Young, Adam L. and Yung, Moti. "Malicious Cryptography Exposing Cryptovirology." Indiana: Wiley Publishing, Inc. 2004.

Appendices

multifileencrypt.cpp

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <mmsystem.h>
#include <malloc.h>
#include <string>
#include <iostream>
using std::cout;


/* ****************************** main() ****************************** */

int main()
{
	int length, numseg; //, nlength, rem, numzero;

	ifstream filesize;
	filesize.open ( "C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\1.wav", ios :: binary );

	//get length of file:
	filesize.seekg ( 0 , ios :: end );
	length = filesize.tellg();
	filesize.seekg (0 , ios :: beg );

if (length % 2048 == 0 )
{
	numseg = length/2048;
}
else
{
	numseg = (length/2048)+1;
}
int end = numseg + 80;
```

```
/*

		//If DataSize is not a multiple of 512 then modify so that it is a
multiple of 512
		//If DataSize % 512 != 0 find nlength to make DataSize % 512 = 0
		if (length % 2048 != 0 )
		{
			rem = length % 2048;					//
remainder of DataSize
			numzero = (2048 - rem);
			nlength = length + numzero;		// new length is
makes DataSize % 512 = 0
			numseg = nlength/2048;
		}

		else
		{
			numseg = length/2048;
		}

		int end = numseg + 80;

		//open binary file in output mode to pad with zeros
		ofstream out;
		out.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\1.wav", ios::out | ios::binary | ios::app);

		out << 1;  // append 1 to end of audio data

		for (int i = 0; i < numzero; i++)  // append "numzero" zero's at end
of file to make DataPortion % 512 = 0
		{
			out << 0;
		}
*/
		//OPEN THE FILE TO BE ENCRYPTED
		ifstream in;
		in.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\1.wav", ios::in | ios::binary );
```

```
		//READ IN CONTENTS OF FILE & STORE IN ARRAY
		ofstream message;

		message.open("C:\\Documents and Settings\\Nathan Inch\\Desktop\\Rhema\\thesis\\message.wav", ios::out | ios::binary );

		//CREATE KEY (ARRAY OF INITIALIZATION VALUES TO BE XORED WITH MESSAGE TO CREATE CIPHER TEXT)
		ofstream key;
		key.open("C:\\Documents and Settings\\Nathan Inch\\Desktop\\Rhema\\thesis\\key.wav", ios::out | ios::binary);

		//OUTPUT ONLY THE ENCRYPTED SEGMENT TO FILE (for debugging purposes)
		ofstream encryptseg;
		encryptseg.open("C:\\Documents and Settings\\Nathan Inch\\Desktop\\Rhema\\thesis\\encryptedseg.wav", ios::out | ios::binary);

		//OUTPUT ONLY THE ENCRYPTED SEGMENT TO FILE (for debugging purposes)
//		ofstream encryptseg2;
//		encryptseg2.open("C:\\Documents and Settings\\Nathan Inch\\Desktop\\Rhema\\thesis\\encryptedseg2.wav", ios::out | ios::binary);

		//OUTPUT ONLY THE ENCRYPTED SEGMENT TO FILE (for debugging purposes)
		ofstream encryptmessage;
		encryptmessage.open("C:\\Documents and Settings\\Nathan Inch\\Desktop\\Rhema\\thesis\\encryptmessage.wav", ios::out | ios::binary);

		//OUTPUT ONLY THE ENCRYPTED SEGMENT TO FILE (for debugging purposes)
//		ofstream encryptfile;
//		encryptfile.open("C:\\Documents and Settings\\Nathan Inch\\Desktop\\Rhema\\thesis\\encryptedfile.wav", ios::out | ios::binary);

		//READ IN VALUES FROM FILE TO BE ENCRYPTED
		ifstream inm;
```

```
        inm.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\encryptedseg.wav", ios::in | ios::binary );


    char segment[480][2048]; //char array to hold message, partial digest,
message digest
        //char decryptmessage[480][1024]; //char array to hold
decrypted message digest, decrypted message segments

    for (int count = 0; count <= 2; count ++)
    {

    if ( count == 1 )
    {
        cout << "COUNT 1: " << count << endl;
        for (int i = 0; i <= end ; i++)  //read in values and create MD and
encrypted portion
        {
            for ( int m = 0; m <= 2047; m++ )
            {
                int t = (i-numseg);

                /*****************
                *MESSAGE SEGMENTS*
                *****************/
                if ( i  >= 0 && i <= numseg )
                {
                    segment[i][m] = in.get();
                    message << segment[i][m];
                }
                // end get message

                /*******************
                *PARTIAL DIGEST/KEY *
                *******************/
                else if ( i > numseg && i <= numseg+40 )
                {
                    segment[i][m] = ((segment[i-50][m/2] ^
segment[i-60][m/5]) ^ segment[i-70][m/13]);
```

```
                                        key << segment[i][m]; //outputs values of
key to file key.wav
                                }
                                // end partial digest


                                //encrypt
                                else if ( i > numseg+40 && i <= numseg+80 )
                                {
                                        //t = i-numseg = (numseg+41)-numseg = 41,
42,43...
                                        //i-40 = (numseg+41)-40 = numseg+1 ...
((numseg+42)-40)= numseg+2 ... numseg+3, numseg+4....
                                        segment[i][m] = (segment[t-1][m] ^
segment[i-40][m]); //encrypt numseg+1
                                        //encryptseg << segment[i][m];
                                        segment[t-1][m] = segment[i][m];  //reassign
values starting at segment (t-1) = 40, 41, 42 to encrypted m
                                        encryptseg << segment[t-1][m]; // segment i
= 40 - 80 are encrypted

                                }
/*
                                //output to file
                                else if (i >= end && i < end + numseg )
                                {
                                        //[i-end]=0,1,2
                                        encryptfile << segment [i-end][m];

                                }
*/
                        }//end increment m

                }//end increment i
                }//count = 1

                else if ( count == 2)  // for debugging ensure that encrypted
portion is being included
                {
                cout << "COUNT 2: " << count << endl;
```

```
                for (int n = 0; n <= numseg ; n++)   //read in values and create
MD and encrypted portion
                {
                        for ( int m = 0; m <= 2047; m++ )
                        {
                                encryptmessage << segment[n][m];

                        }//end increment m
                }//end increment n
                }//count = 2
}//end increment count


                cout << "SEGMENTS = " << numseg << endl;
        return(-1);

}// end main
```

multifiledecrypt.cpp

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <mmsystem.h>
#include <malloc.h>
#include <string>
#include <iostream>
using std::cout;


/* ****************************** main() ****************************** */

int main()
{

	int length, numseg; //, nlength, rem, numzero

	ifstream filesize;
	filesize.open ( "C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\encryptmessage.wav", ios :: binary );

	//get length of file:
	filesize.seekg ( 0 , ios :: end );
	length = filesize.tellg();
	filesize.seekg (0 , ios :: beg );

	if ( length % 2048 == 0 )
	{
		numseg = length/2048;
	}
	else
	{
		numseg = (length/2048)+1;
	}
	int end = numseg + 80;

/*
```

```
		//If DataSize is not a multiple of 512 then modify so that it is a
multiple of 512
		//If DataSize % 512 != 0 find nlength to make DataSize % 512 = 0
		if (length % 2048 != 0 )
		{
			rem = length % 2048;					//
remainder of DataSize
			numzero = (2048 - rem);
			nlength = length + numzero;		// new length is
makes DataSize % 512 = 0
			numseg = nlength/2048;
		}

		else
		{
			numseg = length/2048;
		}

		int end = numseg + 80;

		//open binary file in output mode to pad with zeros
		ofstream out;
		out.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\encryptedfile.wav", ios::out | ios::binary
| ios::app);

		out << 1;  // append 1 to end of audio data

		for (int i = 0; i < numzero; i++)  // append "numzero" zero's at end
of file to make DataPortion % 512 = 0
		{
			out << 0;
		}
*/
		//OPEN THE FILE TO BE DECRYPTED
		ifstream in;
		in.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\encryptmessage.wav", ios::in | ios::binary
);
```

```
		//OUTPUT CONTENTS OF FILE & STORE IN ARRAY
		ofstream encmessage;

		encmessage.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\encmessage.wav", ios::out | ios::binary );

		//READ KEY (ARRAY OF INITIALIZATION VALUES TO BE
XORED WITH ENCRYPTED FILE TO YIELDING MESSAGE)
		ifstream inkey;
		inkey.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\key.wav", ios::out | ios::binary);

		//READ KEY (ARRAY OF INITIALIZATION VALUES TO BE
XORED WITH ENCRYPTED FILE TO YIELDING MESSAGE)
		ofstream key2;
		key2.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\key2.wav", ios::out | ios::binary);

		//READ KEY (ARRAY OF INITIALIZATION VALUES TO BE
XORED WITH ENCRYPTED FILE TO YIELDING MESSAGE)
		ifstream enc;
		enc.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\encryptedseg.wav", ios::out | ios::binary);

		//OUTPUT ONLY THE DECRYPTED SEGMENT TO FILE (for
debugging purposes)
//		ofstream encryptseg2;
//		encryptseg2.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\encryptseg2.wav", ios::out | ios::binary);

		//OUTPUT ONLY THE DECRYPTED SEGMENT TO FILE (for
debugging purposes)
		ofstream decryptseg;
		decryptseg.open("C:\\Documents and Settings\\Nathan
Inch\\Desktop\\Rhema\\thesis\\decryptedseg.wav", ios::out | ios::binary);

		//OUTPUT ONLY THE DECRYPTED SEGMENT TO FILE (for
debugging purposes)
//		ofstream decryptseg2;
```

```
//		decryptseg2.open("C:\\Documents and Settings\\Nathan Inch\\Desktop\\Rhema\\thesis\\decryptedseg2.wav", ios::out | ios::binary);

		//OUTPUT ONLY THE ENCRYPTED SEGMENT TO FILE (for debugging purposes)
		ofstream decryptmessage;
		decryptmessage.open("C:\\Documents and Settings\\Nathan Inch\\Desktop\\Rhema\\thesis\\decryptmessage.wav", ios::out | ios::binary);

		//OUTPUT ONLY THE ENCRYPTED SEGMENT TO FILE (for debugging purposes)
//		ofstream decryptfile;
//		decryptfile.open("C:\\Documents and Settings\\Nathan Inch\\Desktop\\Rhema\\thesis\\decryptedfile.wav", ios::out | ios::binary);



	char segment[480][2048]; //char array to hold message, partial digest, message digest
		//char decryptmessage[480][1024]; //char array to hold decrypted message digest, decrypted message segments

	for (int count = 0; count <= 2; count ++)
	{

	if ( count == 1 )
	{
		cout << "COUNT 1: " << count << endl;
		for (int i = 0; i <= end ; i++)  //read in values and create MD and encrypted portion
		{
			for ( int m = 0; m <= 2047; m++ )
			{
				//int t = (i-numseg);

				/*************************
				*ENCRYPTED MESSAGE SEGMENTS*
				*************************/
```

```
                        if ( i  >= 0 && i <= numseg )
                        {
                                segment[i][m] = in.get();
                                encmessage << segment[i][m];

                        }
                        // end get message

                        /******
                        * KEY *
                        ******/
                        else if ( i > numseg && i <= numseg+40 ) // 40
segments of key.wav
                        {
                                segment[i][m] = inkey.get(); //read in values
from key file created when file was encrypted
                                key2 << segment[i][m]; //outputs values of
key to file key2.wav for debugging (compare to key.wav)
                        }
                        // end partial digest


                        //decrypt
                        else if ( i > numseg+40 && i <= numseg+80 )
//defines # of segments -- 40 segments
                        {
                                int t = (i-numseg);  //41, 42, 43 - 81
                                int l = (t-1);               //40, 41, 42- 80
                                segment[i][m] = segment [l][m];
//                              encryptseg2 << segment[i][m];      // ensure
encrypted seg is exact -- encryptedseg.wav
                                segment[l][m]= segment[i][m]^segment[i-
40][m]; // XOR encrypted section with key
                                decryptseg << segment[l][m];  //segment 40-
80 are decrypted
                        }
/*
                        //output to file
                        else if (i >= end && i < end + numseg)
                        {
```

```
                                        //[i-end]= 0,1,2
                                        decryptfile << segment [i-end][m];

                                }
*/
                        }//end increment m

                }//increment i until last segment needed
                }//count = 1

                else if ( count == 2)  // for debugging ensure that encrypted
portion is being included
                {
                cout << "COUNT 2: " << count << endl;
                for (int n = 0; n <= numseg ; n++)
                {
                        for ( int m = 0; m <= 2047; m++ )
                        {
                                        decryptmessage << segment[n][m];

                        }//end increment m
                }//end increment n
                }//count = 2
}//end increment count


                cout << "SEGMENTS = " << numseg << endl;
        return(-1);

}// end main
```