Generation of Pseudorandom Numbers From Microphone Input in Computing Devices

A Thesis
In TCC 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Science

By

Giles Cotter

3/26/02

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in TCC Courses.

____

Approved____(Technical Advisor)
Professor David Evans

Approved____(TCC Advisor)
Professor I.H. Townsend

Preface

I would like to give my thanks to my two Thesis advisors. Without their advice and assistance this project would most likely have never been completed. My technical advisor, Professor David Evans, provided the technical know-how that stimulated me in the development of my topic, and in its fruition. His advice pointed me in the right direction on many occasions, and redirected me when I went astray. I would like to thank him very much for his advice and support.

Even with a sound topic, I would have been stuck if I could not have successfully communicated my work on paper. For invaluable assistance in the actual writing of the thesis I have to give my thanks to Professor Ingrid Townsend. Her teaching and prompting throughout the TCC 401 and 402 series greatly increased my ability to sit down and write this thesis. I have no doubt that the TCC series will prove a valuable asset throughout my post-UVA career.

Table of Contents

Glossary of Terms and Acronyms

C++ – High level programming language
COE – Common Operating Environment
CS – Computer Science
EE – Electrical Engineering
Entropy – a general term for randomness
GCCS – Global Command and Control System
IT – Information Technology
MB – Mega-Byte – A unit of storage size in computers. 1MB = 2^{20} x 8 bits
PC – Personal Computer
PDA – Personal Digital Assistant
WAV – Waveform Audio File Format

Abstract

Randomness is at the core of the encryption that keeps data secure in the world today. Unfortunately, it is very hard to find good sources of randomness. The aim of this project was to determine if that the computer microphone could be used as a valid source of randomness for use in various applications. The greatest challenge with developing methods of random number generation is ensuring that they are not breakable by someone with perseverance and technological know how. Therefore, the primary goal of this project was to develop a method of random number generation that is very hard, if not impossible, to predict. Second, computing devices are becoming much smaller, and are losing many of the traditional sources of randomness, such as the keyboard. Therefore, the second goal of this project was to develop a random number generation process that can be used on a wide range of computing devices.

To solve these problems, I chose to use the microphone, a feature that is now standard in almost all new PCs and many PDAs. Since each microphone will record sound slightly differently owing to mechanical differences, my hypothesis was that a microphone is a good device from which to gather randomness. The main objective of this project was to develop a method that uses the random sound input to a microphone in the production of random numbers.

Through testing, I have determined that a microphone based random number generator is feasible and that ambient sound is not suitable for the majority of entropy work. Radio static, however, produced very good entropy. Therefore, with a carefully chosen sound source, a microphone can indeed act as a very effective source of randomness.

Chapter 1: Introduction

Encryption has become a necessary precaution when transmitting data from one location to another. However, encryption algorithms are almost always based on a pseudo-random number generator, which is used to start the algorithm. If a person can determine the random number that was used as the "seed" or starting point of the encryption, then that person can easily decrypt the data. Therefore, it is necessary to have forms of random number generation that are not easy to replicate. The aim of this project was to demonstrate that a microphone could act as an effective source of randomness.

The problem that I dealt with for this project was twofold. First, the greatest challenge when developing methods of random number generation is ensuring that they are not breakable by someone with perseverance and technological know how. Therefore, the primary goal of this project was to develop a method of random number generation that is very hard, if not impossible, to predict. Second, computing devices are becoming much smaller, and are losing many of the traditional sources of randomness, such as the keyboard. Therefore, the second goal of my project was to develop a random number generation process that can be used on a wide range of computing devices.

To solve these problems, I have used the microphone, a device that is now standard in almost all new PCs and PDAs. Since each microphone records sound slightly differently owing to mechanical differences, I hypothesized that a microphone may be a perfect device from which to gather randomness. My goal in this project was to develop a method that uses the random sound input to a microphone in the production of random numbers. Through testing, I have determined whether a microphone based random number generator is feasible, secure and useful. As will be discussed in more detail later, my experimentation led me to the conclusion that a microphone based random number generator is indeed feasible. However, there are several restrictions that must be in place in order for this to be true. Ambient noise did not work well at all, whereas radio static was very effective.

1.1 Rationale

This thesis project attempts to show that microphone input could be a valid and successful means of producing random numbers for encryption. My main goal throughout the project was to develop an effective algorithm that produces random numbers from the input to a computer microphone. In order to show that the algorithm I developed is secure, I conducted thorough tests that conclusively determined whether or not the random numbers produced by the algorithm could be easily predicted.

1.1.1 Why is this a relevant topic?

The production of random numbers is key to the successful encryption of data on computer systems. Therefore, easy and secure ways of producing randomness are very useful. Microphones are now standard equipment (or will be in the very near future) on virtually all computers and PDAs. An algorithm that utilizes this resource to produce effective randomness would have great value on a very wide range of computers and computerized devices. Therefore, the main benefit of such an algorithm is its usefulness on an amazing assortment of machines, including those that are lacking in proper security at this moment.

This lack of security is an especially pressing problem on PDAs and other wireless devices. These machines are becoming almost ubiquitous in the business world. Since they are being used more and more for sensitive business transactions, it is essential that effective security measures be implemented. A microphone based random number generator would not be hard to implement on PDAs, since most have the required hardware built in, and the software requirements would be minimal. This type of algorithm would therefore be attractive to PDA software and hardware designers, as it would not require extensive modifications to the existing hardware.

1.1.2 What did this research require?

I first researched the main topics in this area to find out the state-of-the-art. Next, I created an algorithm that takes the sound input from a microphone and transforms it into random numbers. Finally, through experimentation and testing, I proved the viability of the algorithm. The usefulness of the algorithm depended on its ability to produce cryptographically sound random numbers. Now that the project is complete, I have determined the feasibility of a microphone based random number generator.

1.2 Impact Statement

As with any project, it is important to consider the possible impacts and repercussions of the technology that I have implemented. After analyzing the subject matter of my project, and the technical area in which it resides, I have determined that there are two general areas of possible impact: Enhanced computing security and expanded security support. Aside from these major concerns, there are also smaller, more focused areas of impact. These include E-commerce applications, military applications, and social impacts. The following sections detail the possible positive and negative impacts in each of these areas.

1.2.1 Areas of Impact

Enhanced Computing Security

Depending on the algorithm used, encryption algorithms that use the random numbers created by a successful microphone generator could be very secure. This is because the microphone will produce randomness that is very hard, if not virtually impossible, to reproduce. This will be a positive impact in that it will increase the safety and effectiveness of computer security as a whole, by offering a new alternative to the current methods of random number generation. Many methods used presently have proved to be easy to defeat or replicate.

There could be negative impacts to computer security, however. If microphone based encryption was adopted, there is the possibility that in the future a new technological advancement would make it possible to successfully duplicate the randomness produced by the algorithm. This would cause a huge problem, especially if the method was in wide usage. Since the microphone is such a commonplace integrated device, there is a strong possibility that a vast number of devices could be using such random number generation, and thus would be vulnerable should such a flaw be found.

Expanded Security Support

The introduction of an effective microphone based generator would enable the easy production of random numbers on a huge variety of computing devices. This would basically include any device with an integrated microphone and sound processing technology built in. Because of the wide availability of microphones in new computing devices, it will be possible to make secure transactions from devices that previously were unavailable, or unsafe to use. This is obviously a very positive impact, as a high level of security would become available to a hugely expanded array of computing devices.

E-Commerce Applications

E-commerce from wireless PDAs would be facilitated by the introduction of a microphone based random number generator. Current PDAs, while having quite advanced wireless communications, are thoroughly lacking in security. This is one factor that makes online purchasing a risky undertaking with the current level of technology. However, increasingly PDAs are being equipped with integrated microphones and basic sound processing hardware and software. This is all that would be needed to implement a microphone based random number generator to be used for encryption. Successful random number generation would therefore enable a surge in E-commerce, since PDAs are almost ubiquitous items these days. This gives online retailers the opportunity to obtain customers even when the customers are not in their office or at their home PCs. With the success of microphone-based encryption, the time frame in which people could make online purchases would be expanded greatly. This is a positive impact not only for the retailers themselves, but the economy as a whole.

Military Applications

The military is currently using a Global Command and Control System (GCCS) that allows the dynamic tracking of military units all over the globe. During my internship at the Science Applications International Corporation, I was exposed to an extension of this system. This extension would allow units all over the globe to view positioning information in real time on small wireless computing devices, such as pen tablet computers. One of the main problems with the development of this system is the security of the transmissions, since the data to be transmitted is very sensitive. As these are small devices, not many traditional methods of random number generation will be available. However, microphones are becoming standard equipment in the vast majority of computing devices, even those smaller than the ones that would be used in this case. Therefore, a microphone based random number generator could be a very useful and secure method of generating keys for the encrypting of this tactical data before it is wirelessly transmitted.

There are, however, two possible long-term negative impacts. First, should the encryption method come to be used by the military, and then a flaw be found, national security could be at risk, since anyone possessing the means to replicate the random numbers would be able to crack the encryption used by the military devices. This could have terrible consequences, especially if the devices are sending classified or battlefield tactical data. Fortunately, this scenario is not very likely since the military is constantly upgrading its technology, and would most likely discover any flaws in the algorithm before would-be troublemakers. Second, military enemies could possibly adopt this method to obtain higher level of encryption security. This would add to their strength and effectiveness. This is unfortunately a reality in the world today. However, microphone-based encryption would not necessarily increase the strength of the encryption, but rather provide a simple source for key generation. Since the United States recently lifted the export restriction on powerful 128-bit encryption, I do not believe that the availability of a microphone based random number generator would compromise the security of the United States military, as powerful cryptographic algorithms are already widely available.

Social Impact

On a psychological note, the ability for people to make easy and secure transmissions from wireless PDAs anywhere in the world would mean that the wired world is encroaching even more on every aspect of life. While traveling, a person used to be isolated from the hustle and bustle of the business world. However, with effective encryption methods for even such small devices, a person would no longer be able to get away from work or the tech world in general. This may cause a rise in stress, and other psychological problems caused by lack of rest or working too hard. For instance, even when a businessperson is on vacation in this day and age, he or she is rarely without a cell phone, pager, or computer. Therefore, the relaxation of the vacation time is lost, since the office is able to interrupt at any moment. Stress levels will obviously rise as a result. This can be seen as an unfortunate, but inevitable, impact of the rapid advancement of technology as a whole.

1.2.2 Overall Impact Assessment

After carefully weighing the potential positive and negative impacts of this project, I have come to the conclusion that the possible gains are worth the risks involved. The advancement in personal, business, and military technology far outstrips any of the losses that may occur in the long-term. An increase in overall computing security, as well as a hugely expanded variety of secure devices would, in turn, bring great benefits to the personal, business and military worlds. Therefore, based on this analysis, I believe that a microphone based random number generator was a worthwhile and socially responsible project to pursue.

1.3 Report Structure

Chapter 2 presents background information that is necessary for a full understanding of the subject matter being dealt with in the project. This chapter also discusses the state-of-the-art in the area of random number generation and encryption. Chapter 3 discusses the process that went into the creation of the algorithm used in the production of the data for this project. The actual listing of the algorithm code used can be found in the appendices at the end of this report. Chapter 4 discusses the test suite that was used to evaluate the data collected, and the actual sound samples that were obtained. Chapter 5 describes the results that were collected and gives an analysis of the implications of this data. As the full data set is very long and unreadable, only simple descriptions of each file are included in the appendices to this report. Finally, Chapter 6 concludes the report with a summary of results, and a general statement as to the success or failure of the project.

Chapter 2: Background

This Chapter gives the reader the relevant background information that is needed to have a better understanding of the subject matter covered in this report. It also discusses the state of the art in the area of random-number generation and encryption.

This section of the thesis report gives background information and a brief overview of important aspects of random numbers and their uses in the modern world. This will give the reader a clearer picture of the true worth of the project. Since the rest of the project report deals only with the production of random numbers and not their possible applications, this chapter also provides a broader view of the uses of randomness in technology today.

2.1 Background Information

2.1.1 Random Numbers

What is commonly referred to as a "random number" in the computer science world is actually not truly random. Due to the non-random nature inherent in computers, it is virtually impossible to generate real randomness, and so, the numbers that are produced by computer generators are actually "pseudorandom" numbers. Pseudorandom numbers are "numbers that are generated from some (hopefully random) internal values, and that are very hard for an observer to distinguish from random numbers"[8:1]. Pseudorandom numbers are of great importance in cryptography, as they are used to generate keys for encryption algorithms, and nonces for authentication protocols. Unfortunately, many of the internal values available in computers are in no way random enough. The system clock, which is often only accurate to milliseconds, is the classic example of a bad source of randomness. Surprisingly, the system clock is actually used in many programs. These programs are not secure, as the system clock is very predictable, and it is not hard for an attacker to ascertain the pattern being used. This reveals the main weakness of pseudorandom numbers. If the sources of entropy, or randomness, are not random enough, then systems using these pseudorandom numbers for encryption will be very vulnerable.

2.1.2 Key Based Encryption

The main use of random numbers in the tech world today is in encryption algorithms. Encryption is the process by which data is protected by being transformed into 'ciphertext' that can only be understood by an authorized person [1]. Manual encryption has been used ever since the time of the Romans, but it is now most commonly used on computer systems to send data securely. A widely used form of encryption is symmetric key based encryption. In the most basic form of key based encryption, the sender encrypts the data using the encryption algorithm with a special "key" number plugged into it. On the receiving end, the same key must be used in order to decrypt the data [1]. A variation, known as public-key encryption, uses two keys. In this case, one public key is used to encrypt data, and another private key is used to decrypt it on the other side [2]. These algorithms use random numbers to produce keys. In order to maintain secure encryption, it must be very hard, if not impossible for an unauthorized user to guess the correct key values. Therefore, the numbers used as keys must be as close to truly random as possible. This emphasizes how important it is that the microphone based random number generator produce numbers that are virtually impossible to replicate.

2.1.3 WAV File Format

In order to create randomness from sound, one must store the data in a format that is easily modifiable and accessible. One of the simplest forms of sound file is the WAV file, or Waveform Audio File Format [7]. This file format was developed by Microsoft, and has been supported by Windows for many years. The main advantage of the WAV format, with respect to this project, is that the sound data is stored in a completely raw format. In other words, no compression or manipulation is performed on the sound data as it is recorded. This is obviously not ideal when disk space is a concern. A three to four minute WAV file can take up to 50MB of space, whereas other formats that use compression take only about 3MB for the exact same data.

In this project, however, this raw data storage is ideal, as it means the data stored is pure and unmodified. This allows for the direct and easy manipulation and analysis of the sound data. The data itself is stored in a series of chunks called bytes. Each byte consists of 8 binary bits. Each of these bits can be either 0 or 1, meaning there are 256 possible permutations in each byte. The information about the file, including the sampling size, length, and number of channels, is stored in a header at the beginning of the file. The data simply follows the header as a sequence of bytes. A detailed breakdown of the header and data of the WAV file format can be found in Appendix C.

2.2 State of the Art

2.2.1 Yarrow-160 and Tiny

One of the best examples of a current random number generator is the Yarrow Cryptographic Pseudorandom Number Generator [8]. Designed by Counterpane Systems, this pseudorandom number generator is very effective at producing pseudorandom numbers that are as close to being truly random as possible. To accomplish this, Yarrow was designed from the ground up to compensate for the flaws in earlier generators, and to compensate for the types of possible attacks that a malicious user might launch against the system [8]. Yarrow can collect entropy from a variety sources in the host system, and is very effective at making the most of this randomness. One of the most successful implementations of the Yarrow standard is the Tiny Pseudorandom Number Generator, and the related Tiny Encryption Algorithm [6]. Tiny has empirical test data that supports the claim that the Yarrow specification is capable of very good random number generation. However, once again, the true effectiveness of the system is determined by the randomness of the sources provided to it. As is stated in the Yarrow documentation, "implementors should be very careful in determining their sources. The sources should not be closely linked or exhibit any significant correlations"[8:7]. In other words, even the very effective Tiny generator can be vulnerable if the inputs to the generator are not random enough. This, once again, highlights the importance of the randomness of the sources of entropy in random number generators. Even the most advanced and sophisticated generator can theoretically be cracked if it has bad sources of randomness. This project will aim to prove that microphone input is a very good source of randomness for use in such generators.

2.2.2 Random.org

Random.org is a web site that has been very successful at using sound to produce randomness [4]. The creator of the web site, Mads Haahr, has created a random number generator that uses a microphone and processing software to produce very good pseudorandom numbers. The basic approach that Haahr used was to feed environmental static into a Sun computer through a microphone. This recorded data was then fed through manipulation software, which transformed the sound data into usable random data. The randomness produced by the Random.org generator is streamed continuously, and can be linked to directly from other web pages. In this way, other web page administrators who require random numbers can simply draw from the Random.org stream. Overall, it is a very well organized operation, and adds a lot of credibility to the hypotheses of this project. Random.org was an invaluable aid in the design and testing of my processing algorithm and data. In fact, I first discovered the randomness test suite I used to test my data, ENT [5], on Random.org. This is the test suite that Random.org uses to evaluate the value of all the numbers it produces. The effective evaluation of entropy, and the actual workings of the ENT test suite are discussed in detail in Chapter 4.

2.2.3 Biometrics - Key Frequency and Voice Based Key Generation

Aside from the obvious application in encryption, there are many other uses of randomness being researched today. The ultimate goal of these research projects is to find a source of randomness that is completely impossible to predict or duplicate. The sources that are currently being examined range from pen input into PDAs to fingerprints and retina scans. The basic premise behind all of these new approaches is to use the natural randomness that is inherent in the person using the system. The use of these inherent human qualities in computer security is known as biometrics. Two promising biometric sources of randomness are keystroke frequency [10] and the human voice [9].

Keystroke frequency techniques base their randomness on the rhythm with which a user types. As Fabian Monrose states, when a person types it is possible to track the exact latencies associated with that person's typing style. Through this analysis, it is possible to construct a unique "signature" of sorts that can be used to positively identify the individual typing [9]. Therefore, if the rhythm with which one types is unique, it makes sense that this uniqueness could be used as the key to an authentication program. This is closely related to key based encryption programs which also use keys as a form of authentication.

The second, and more recent, research deals with the human voice. In this case researchers are attempting to use a spoken password, rather than keyboard measurements, to authenticate a user. This is an even more promising source, as a voice can have more inflections and variances than even keyboard rhythm [9]. It is easy to see that my project is related to this research, as microphone generated keys are involved. However, rather than using a specific voice to match a password, I will be using a microphone to gather random noise, and will then translate that into pseudorandom numbers.

The research into biometrics can be seen as the opposite end of the spectrum of randomness experimentation, with respect to this project, as it deals more with repeatable randomness inherent in a human being. This randomness should not be able to be reproduced by anyone other than the human that produced it in the first place. However, the main point is that it is repeatable. This project is aimed at producing randomness, with a microphone, which cannot be repeated ever.

Chapter 3: Algorithmic Design and Description

This Chapter discusses the process that went into the creation of the algorithm used to transform sound data into pseudo-random numbers. It also presents a step-through of the main functionality of the code.

The main standard that I set forth when designing the processing code for this project was that I should aim to modify the data as little as possible. I wanted to see how much modification was necessary to achieve good randomness. A description of exactly how I judged which randomness was good can be found in Chapter 4; this is where the data and test suite are detailed in full. As it turns out, the modifications that were necessary to achieve acceptable levels of entropy were minimal, but the coding was far from simple. The alterations necessary for this project required direct access to the individual bits of the data. In my experience, C++ is not designed to deal with data at the bit level effectively. This proved to be the most challenging aspect of the coding for the project. The following sections describe the processing program at a high level. In depth coding minutiae are not necessary to understand what the program does, and are therefore not detailed. A full listing of the code can be found at the end of the report, in Appendix A.

3.1 Program Input

The data for this project was composed of a series of WAV files recorded with a computer microphone under various conditions. As described in Chapter 2, a WAV file consists of a descriptive header followed by a stream of sound sample bytes. The header file is very structured, and therefore would detract from any randomness in the file. So, instead of simply feeding a WAV file directly into my processing software, I first manually strip the header off the file. This is as simple as opening the file in a text editor and deleting the header bytes as described in the WAV specification (Appendix C). Originally, I had planned to build this header removal into the processing software. However, I discovered that the length of the WAV header actually varies slightly from file to file. Therefore, I could not generalize the removal into a section of code, and had to resort to manual deletion. Once the header is truncated, the remaining file consists of a stream of raw sound data, and is ready to be plugged into the processing software. This raw data file, therefore, is the required input for the processing software.

3.2 Program Functionality

After preparing the input data file, as described above, the processing software can be run on it. What follows is a simple breakdown of the main functionality of the processing code. It will become obvious that the actual modifications performed by the code are not extensive. This is in keeping with my philosophy of making minimal changes to achieve acceptable levels of entropy. Furthermore, as discussed earlier, the main challenge in writing this code was not the modifications themselves, but rather dealing with the data at the bit level using a high level programming language.

The core action of the program is the removal of the high seven bits of every data byte. Recall that each byte consists of eight bits. Basically, every data byte is read in one by one, the first seven bits encountered are removed, and the remaining eighth bit is written to the output file. When the processing is complete, therefore, the output file consists of the concatenation of the eighth bits of every byte in the original input data file. The reasoning behind this is simple. The high seven bits of every byte will quite often be identical. Leaving these duplicate bits in will severely lower the entropy present in the output file. As well as being suggested by my technical advisor, this truncation of bits is one of the first actions taken by the software behind the very successful Random.org generator (see Section 2.2.2). The validity of this approach can clearly be seen in the results of this project, as will be described in Chapter 5.

When I was first wrote the code for the processing software I also had a routine that removed “silence” characters from the data before processing it. A silence character is a sequence of bits inserted by the recording program when the microphone is not picking up any sound at all. Through experimentation, I found that the silence character consisted of the eight bit sequence “10000000.” Therefore, my initial code made a pass through the data and removed all bytes containing this sequence. However, when I eventually reached the testing phase, I soon found that I could not achieve good entropy with the silence characters removed. After taking out the removal code, the entropy increased dramatically. Therefore, the final version of the code, as listed in Appendix A, does not remove silence characters.

3.2.1 Program Breakdown

Setup Phase:

- Prompt the user to enter the input filename of the raw data file to be processed. This is stored so that the software knows where to look when it is fetching the input data.

- Prompt the user to enter the filename of the output file to create. This filename is stored and used later when output is being written.

- Open the raw data file, using the input filename provided by the user, and copy the sound data into a buffer where it will be easily accessible during the remainder of the program.

- Report the total number of bytes collected to the user.

Processing Phase:

- Remove the high 7 bits of every byte. To do this loop through each individual byte, access the binary data, and only store the 8th bit. The other 7 bits are simply discarded.

- Save the 8th bits of every byte, ready to be written to the output file.

Output Phase:

- Concatenate the collected 8th bits together into the final output form.

- Write the output data to a filename and location as given by the user in the setup stage. This file will then be accessible to the user for testing.

- Inform the user that processing is complete.

- Perform final clean up, and exit.

3.3 Program Output

What follows is a sample run-through of the program to illustrate the user prompts, and the activities required of the user. As I had to run this processing script for every WAV file I recorded, I endeavored to make it as simple and fast as possible. Therefore, the program requires a minimal amount of typing and interaction, while still offering flexibility. It should be noted that the program pauses on several occasions to allow the user to enter information, or to give the user time to read the information presented on the screen.

The final output of the processing program is a data file containing a modified stream of bytes of information. This file is saved in the location specified by the user during program execution. This file is completely ready to be examined by the testing suite, and no further modification is required. The actual testing procedures are discussed in Chapters 4 and 5.

3.3.1 Sample Program Execution

```
|~~~~~~~~~~~~~~~~~~~~~~~|
| WAV Processing Script |
|-----------------------|
|                       |
| Author:  Giles Cotter |
|                       |
 ~~~~~~~~~~~~~~~~~~~~~~~


Please enter the WAV data file to process: ./data/base/base.bin

             The input file to be used is './data/base/base.bin'.


Please enter the output data file to save to: ./output/test.out

             The output file to be used is './output/test.out'.


Press any key to begin processing...


The complete file is in a buffer.
Number of characters collected = 1323000.
Therefore, this file consists of 10584000 bits.

Press any key to continue...


# of data bytes available: 1323000
# of bytes used for lowest order bit: 1323000 (divisible by 8)
Therefore, there will be 165375 bytes after processing.

Press any key to continue...


Processing complete.
```

Chapter 4: Data and Test Suite Description

This Chapter discusses the testing procedure used in obtaining the data collected during the course of this project. It also explains the test suite that was used to evaluate the level of randomness present in this data.

4.1 Data Collection

Once I had completed the processing software, I was ready to obtain the data needed to prove the feasibility of a microphone-based system, and proceeded to record every sound file with the standard Windows Sound Recorder 32, in WAV file format. Sound Recorder has a maximum recording length of one minute, and so each data file was allowed to record for the maximum one minute. I recorded in 8-bit mono sound at a sampling rate of 22.05kHz. To give myself a control case I recorded one WAV file with the microphone disconnected. This is the equivalent of recording in a completely silent room. For the remaining data, I decided to attack the problem in two different ways. The first set of recordings I made consisted of ambient room noise, with the microphone in varying positions in the room. This was to establish whether a standard computer microphone could actually draw enough sound from a room with only small sounds such as typing or low talking occurring. The second set of recordings were of various radio frequencies with nothing but static on them. This was more in following with the example of Random.org, and was meant as an alternate method in case ambient sound failed to produce the required levels of entropy. As will be discussed in Chapter 5, the standard computer microphone turned out to be very insensitive, and thus severely limited the usefulness of ambient noise, but performed much better on a direct feed of static from a radio. The following table gives a summary of the data samples collected.

4.1.1 Table of Data Collected

Note: files are listed after header has been removed, and are therefore listed as *.bin* rather than *.wav* files.

Data File Name	Location Recorded/FM Station used
Base.bin	Recorded with microphone unplugged (control)
Ambient1.bin	Recorded directly in front of computer
Ambient2.bin	Recorded with microphone on sitting computer case
Ambient3.bin	Recorded with microphone up on desk
Ambient4.bin	Recorded with microphone sitting in the middle of the room
Ambient5.bin	Recorded with microphone in door to adjoining room
Radio1.bin	Recorded with radio set to 87.5 FM
Radio2.bin	Recorded with radio set to 89.0 FM
Radio3.bin	Recorded with radio set to 98.4 FM
Radio4.bin	Recorded with radio set to 99.6 FM
Radio5.bin	Recorded with radio set to 102.8 FM

Table 1: A Listing of Data Files

4.2 ENT: A Pseudorandom Number Sequence Test Program

To be able to judge whether or not the data I collected had a high level of entropy, I needed an easy to use set of tests that would give me clear results. After some research I discovered that there are several tests used to determine the entropy content of a data sample. Five of the most widely used tests are the information density test, the Chi-square test, the Arithmetic mean test, the Monte Carlo value for Pi, and the Serial Correlation Coefficient. The calculation in some of these tests can become very complicated. Luckily, a freeware program entitled ENT [5] performs these five tests on a data file, and prints out easy to understand results. This is the test suite tool that Random.org uses to evaluate all of the random numbers that they produce. The following section presents an example output from the ENT program, and explains each of the tests in detail.

4.2.1 A Step-by-Step ENT Example

Here is a sample of the output that the ENT program returns for a data file. This is the example given on the ENT documentation web site [5].

```
Entropy = 7.980627 bits per character.

Optimum compression would reduce the size
of this 51768 character file by 0 percent.

Chi square distribution for 51768 samples is 1542.26, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 125.93 (127.5 = random).
Monte Carlo value for Pi is 3.169834647 (error 0.90 percent).
Serial correlation coefficient is 0.004249 (totally uncorrelated = 0.0).
```

- <u>*Data Density*</u>*:* The first two results from the ENT printout discuss the data density of the data file. The *Entropy* of the file is listed as bits per character figure ranging from zero to eight. Since there are eight bits in one character, or byte, the maximum number of bits of entropy one can hope for is eight. Obviously, the higher the number of bits of entropy per byte of data, the more random the file should be. Data becomes "denser" with higher entropy per byte. The second line discusses the *optimal compression* of the data. A file that is very dense, and therefore has high entropy per byte, cannot be compressed into a smaller size. Therefore, the lower the figure given for optimal compression, the denser the data is, and the more likely it is that it is close to being random. In the example, the Entropy is listed as a high 7.98 bits per byte, and optimal compression would reduce it by 0%. Therefore, the data density tests would seem to indicate that this is a sample with high randomness. However, this is not enough to conclusively say that the data is unpredictable.

- *Chi-square test:* This is one of the most widely used tests for randomness, and is extremely sensitive. The inner calculations of the test are complex, and therefore not listed here. For the purposes of this project it is sufficient to understand the general workings of the function. In basic terms, the Chi-squared distribution is calculated for the data stream, and from this is derived a percentage that indicates how often a truly random sequence would exceed this value. As can be seen in the example above, both the Chi-square distribution and related percentage are listed. In order to judge the randomness of a data file, one simply has to know how to interpret the resultant percentage. According to the ENT documentation, any file that results in a percentage less than 1% or greater than 99% is almost certainly not random. Percentages between 99%-90% and 1%-10% are considered suspect, and are most likely not random. The general rule is that a file has good levels of entropy if its Chi-square percentage ranges from 10% to 90%. Note that while the example had very dense information, it had very poor Chi-square percentage of 0.01%. This indicates that the file is most likely not random.

- *Arithmetic Mean:* This is one of the simplest tests. The arithmetic mean is calculated by dividing the sum of the bytes in the data file by the length of the data file. A file that is close to being truly random has an arithmetic mean that is very close to 127.5. In the example, the data file has an arithmetic mean of 125.93. This value is quite far from 127.5, and so this file is probably not random.
- *Monte Carlo Value for Pi:* In this test, each sequence of six bytes in the data file is considered to be 24-bit X and Y-coordinates in a square. If the distance of this coordinate is less than the radius of a circle inscribed in the square, then the six-byte sequence is considered a "hit." The percentage of hits for the file is calculated, and from this a value for Pi is derived. The closer a file is to being truly random, the closer this derived value will be to the real value of Pi. In the example, the derived value is off by almost 1%, which is not considered to be very random.
- *Serial Correlation Coefficient:* Finally, the serial correlation tells how closely related one byte is to the preceding byte. A non-random file will have a high correlation between each byte. Therefore, in order to be close to truly random, the coefficient should be as close to zero as possible. According to the ENT documentation, a completely non-random C++ file would have a value approaching 0.5. The low value of the serial correlation coefficient in the example would seem to indicate good randomness. However, previous tests have shown this not to be true.

Overall, therefore, none of the individual tests in the ENT suite can conclusively prove that a file is random. However, the combination of all of them can give a very good indication in one direction or the other. Through my background research I found that the hardest question to answer when considering entropy is its quality. The ENT test suite is certainly not the be all and end all of entropy tests. However, I have confidence in saying that it was a very good test suite to use in the testing of my data. It provided an acceptable level of rigor for the scope of this project. The tests that I actually performed and the results are described in detail in Chapter 5.

Chapter 5: Analysis of Results

This Chapter discusses the actual results of the project. It presents an analysis of the data, as well as the output of the test suite. It draws conclusions about these results and makes concrete statements about the implications of these outcomes.

In order to see what the minimal level of modification is to draw sufficiently random numbers from the microphone data I gathered, I ran the data through the ENT test suite both before and after running it through my processing script. In this way, I could observe whether the data coming directly from the microphone was random, or whether it required processing. Therefore, for every data file I have the output of the ENT program in both unprocessed and processed form. Since these results take up quite a lot of space, they can be found in Appendix B. The following sections summarize the ENT results for Unprocessed and Processed data. These are followed by an analysis of what exactly these results mean in the context of the goals of this project.

5.1 Unprocessed Results

Base Case:

The results obtained for the base case in unprocessed form were unsurprising. This file consists of nothing but silence, and is therefore a single "silence character" (10000000) repeated over and over again. Therefore, it is obviously completely non-random. The ENT tests prove this to be correct. All five of the tests gave abysmally bad results. This is a good basis for comparison, though, as it is the exact opposite of the results that I should aim to receive on the real data samples.

Ambient Sound:

The ENT results for all of the unprocessed ambient sound data files were only slightly better than the completely non-random base case. On the most important test, the Chi-square test, they all ended up with a very non-random 0.01%. Also, the bits of entropy per byte never exceeded two bits per byte. It was obvious from these test results that without processing the ambient sounds that a standard computer microphone can record are simply not random in the least. At this point in the testing, however, I was still optimistic that after processing the entropy would increase dramatically.

Radio Static:

The unprocessed radio static data files produced much better results from the ENT test suite. All of the files exceeded four bits entropy per byte, and some even approached six. The arithmetic mean and the serial correlation coefficient were definitely heading in the right direction even if they were not quite random yet. Only the most sensitive test, the Chi-square test once again produced very low scores. However, I was nonetheless pleased with the test results for the unprocessed data set. While the data was obviously not random enough yet, it was definitely close. I was almost certain that after processing I would see very good randomness emerging.

5.2 Processed Results

Base Case:

The results for the processed base case were just as unsurprising. The processing script simply turned the file into a stream of all zeros. This does not even have the benefit of the repeated ones in the original file. Needless to say, the ENT test suite conclusively reported that there was no randomness at all present in this file. Once again, however, this was that aim of this file, being a control. It again provided the situation I wished to avoid in all the other data files.

Ambient Sound:

The ENT tests for the processed sound test were frankly rather disappointing. While the entropy bits per byte did increase slightly, the values were still nowhere near acceptable levels. The Chi-square test remained steadfastly at 0.01% for all of the samples. All of the other tests simply added weight to the conclusion that the ambient sounds recorded simply did not have enough entropy in them. Therefore, I had to conclude that either ambient sound does not contain enough noise to be valid, or my recording process was flawed somehow. More about this can be found in the following analysis section.

Radio Static:

The ENT test suite results for the processed radio data were extremely good. Every single one of the data file produced test results indicative of very good randomness; every file had 7.998 or more bits of randomness per byte, which is extremely dense. As a result, optimal compression could not reduce any of them at all. The arithmetic mean was very good for all of the files, and was almost exactly at 127.5 for three of them. The Monte Carlo Value for Pi was very close to the actual value of Pi in all of the files, even achieving 0.04% error in one of the files. The serial correlation coefficient was extremely close to zero in all cases. Finally, and most importantly, the Chi-square test produced percentages in the acceptable area for all of the files. In fact, three of the files produced a percentage of 50%, which is extremely good for the Chi-square test. Therefore, the ENT test suite showed conclusively that the processed radio data files did contain very high levels of entropy.

5.3 Analysis of Results

With the data fully collected and tested by the ENT test suite, I can now move on and analyze the implications of these results. First I will explain the unsuccessful half of the data set. The ambient noise data files turned out to contain very little entropy even after running through my processing script. I believe there are several possible causes for this. It is probable that the level of ambient noise that was present in the test locations simply was not loud enough to provide sufficient randomness to the microphone. This brings to light the main problem with using ambient noise for randomness: The amazing variance. The ambient noise in my room is vastly different from the ambient noise in a train station, for instance. Therefore, the actual randomness that one could achieve from ambient noise would definitely depend on the current location. This is quite a disadvantage, as most locations do not provide loud ambient noise. I also believe that the standard (and cheap) computer microphone being used was simply not sensitive enough to pick up the quiet ambient noises in the room. Therefore, with an expensive and very sensitive microphone, I have no doubt that sufficient randomness could be drawn from even the quietest of rooms. However, the microphones available in the vast majority of computing devices will be of the same, or even lesser, sensitivity as the microphone I used for my experiments. Therefore, I have come to the conclusion that ambient noise will not be a very good source of entropy in the majority of situations.

While I was obviously disappointed with the results of the ambient noise tests, all was not in vain. The processed radio static tests produced very satisfactory levels of randomness. I believe that the main advantage that these files had over the ambient sound files was that the microphone was fed a constant stream of sound at a level that was easily within the sensitivity bounds of the rather insensitive microphone. Also, there are millions of electromagnetic signals causing the static. It is almost inconceivable that the composition of all of these signals would not be highly random. Therefore, I believe that this half of the data set validates the goals and hypotheses that I set out at the beginning of this project. My goal in this project was to show that successful randomness could be produced using a standard computer microphone with a minimal amount of modification to the sound data. As I received very good entropy from the radio static, and only had to remove the high seven bits of each byte to achieve it, I have no qualms in saying that I have succeeded in my goals. The microphone is definitely a feasible source of randomness. The only caveat is that one has to be careful about the sound source that one uses. Also, radios are not standard equipment in most computers and PDAs at present. This would limit the usefulness of microphone-based encryption. However, radios are cheap, and static can be obtained anywhere in the world. Overall therefore, with careful selection of source, even a cheap computer microphone can produce very good random numbers.

Chapter 6: Conclusion

This Chapter sums up the entire project, giving a brief summary of the material covered in the previous chapters. It then draws final conclusions about the success of the project as a whole.

The goal of this project was to determine if a computer microphone could be used as a very effective source of entropy. The collection and usage of randomness is an essential part of a wide range of computing activities, not least of which is data encryption. There are many successful random number generators in use already, but every one of them depends on a good source of entropy. Therefore, any new successful source of entropy is valuable. A successful microphone based encryption algorithm would also be valuable since the microphone is becoming an almost ubiquitous accessory in the multitude of computing devices available today. It would provide an easy source of entropy to these devices without hardware modifications.

With this goal in mind, I set about designing an algorithm that would make minimal changes to sound data files and yet yield output with very good entropy. I then recorded both ambient noise and radio static to compare the entropy levels occurring in each. Using a test suite of five of the most common tests of entropy, I evaluated the entropy of all of these data files. When the testing was complete, the results were mixed.

Ambient sound turned out to be a less than ideal source of entropy. All of the data files produced results that indicated low levels of usable entropy. I have attributed this to a combination of poor microphone sensitivity and low sound levels in the testing area. With a more sensitive microphone or a louder testing area, I believe good randomness could be achieved. However, this data set revealed that a system based on ambient noise would be very dependent on the recording location and the quality of the microphone.

Since most areas would not be loud and most microphones would not be of high sensitivity, I have to conclude that ambient noise would generally not be a good source of randomness.

Radio static proved to be a much better source of entropy. This data set produced consistently excellent levels of entropy. This is because the sound being fed into the microphone was of a constant volume level and could be carefully controlled. Obviously this is a much better way to obtain randomness through a microphone. While the initial ambient sound experiments were unsuccessful, the radio static data files proved that the microphone is indeed a feasible source of random numbers.

Overall, I believe that this project has been a success. While not all of my data files achieved the levels of entropy that I wished, it did clarify the situation for me. The fact that the radio data files did produce good randomness validated my initial assumption that the microphone can be used to gather entropy. Therefore I did succeed in proving the hypothesis of this project. Looking back over the course of the project I can draw several general conclusions. First, a microphone can be used to gather entropy very effectively. However, in order to gather this entropy one must be very careful about the sound input fed to the microphone. Unless a sufficient volume and consistency is present, a standard microphone will not be able to distinguish enough usable entropy from the silence. So, this project was a success in that it proved the feasibility of a microphone based random number generator, and it provided valuable lessons that can be applied to any future work I may perform in this area of continuing research.

Bibliography

Web Based Resources:

1. "data encryption" *Encyclopedia Britannica Online*.
   <http://search.eb.com/bol/topic?eu=2226&sctn=1>
   [Accessed October 2001].

2. "cryptology" *Encyclopedia Britannica Online.*
   <http://search.eb.com/bol/topic?eu=28529&sctn=1>
   [Accessed October 2001].

3. Litterio F., *The Mathematical Guts of RSA Encryption.*
   <http://world.std.com/~franl/crypto/rsa-guts.html>
   [Accessed October 2001].

4. *Random.org – True Random Number Service.*
   <http://www.random.org>
   [Accessed December 2001].

5. *ENT – A Pseudorandom Number Sequence Test Program*.
   <http://www.fourmilab.ch/random/>
   [Accessed December 2001].

6. *The Tiny Encryption Algorithm (TEA).*
   <http://vader.brad.ac.uk/tea/tea.shtml>
   [Accessed December 2001].

7. *WAV File Format Description*.
   <http://www.technology.niagarac.on.ca/courses/comp630/WavFileFormat.html>
   [Accessed December 2001].

Papers and Reports:

8. Kelsey J., Schneier B., Ferguson N. *Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator*. Sixth Annual Workshop on Selected Areas in Cryptography, Springer Verlag, August 1999.

9. Monrose F., Reiter M., Li Q., Wetzel S., *Cryptographic Key Generation from Voice*. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, Lucent Technologies, New Jersey, May 2001.

10. Monrose F. and Rubin A. *Authentication via keystroke dynamics*. In 4th ACM Conference on Computer and Communications Security, April 1997.

11. Jermyn I., Mayer A., Monrose F., Reiter M., and Rubin A. *The design and analysis of graphical passwords*. In Proceedings of the 8th USENIX Security Symposium, August 1999.

12. L'Ecuyer P., *Uniform Random Number Generation*, Annals of Operations Research 53 (1994), 77-120.

13. Entacher K., *A collection of selected pseudorandom number generators with linear structures*. Technical Report 97-1, ACPC - Austrian Center for Parallel Computation, University of Vienna, Austria, 1997.

14. Kelsey J., Schneier B., Wagner D., and Hall C., *Cryptanalytic attacks on pseudorandom number generators*, Fast Software Encryption, Fifth International Proceedings, pp. 168-188, Springer-Verlag, 1988.

15. Hoover D. and Kausik B., *Software Smart Cards via Cryptographic Camouflage,* {IEEE} Symposium on Security and Privacy, pp. 208-215, 1999.

16. Schneier B., Kelsey J., Whiting D., Wagner D., Hall C., and Ferguson N. *Twofish: A 128-Bit Block Cipher*. In *Selected Areas in Cryptography '98*, June 1998. Lecture Notes in Computer Science (these proceedings).

17. Rogaway P. and Coppersmith D., *A Software-Optimized Encryption Algorithm, Fast Software Encryption*, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 56-63.

18. Gilboa N., *Two Party RSA Key Generation*, Proceedings of Crypto '99, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, pp. 116--129, 1999.

19. Wagner D., Schneier B., Kelsey J., *Cryptanalysis of ORYX*, unpublished manuscript, 4 May 1997.

Appendix A: Listing of Program Code

Note: Microsoft Word is not the ideal format for displaying C++ code, and so some reformatting has had to be done to make the code as readable as possible. Therefore, this code may contain errors and bad formatting if it is simply pasted into a C++ compiler. The original working .cpp file is available at request.

```
#include <fstream.h>
#include <iostream.h>
#include <bitset>
#include <string.h>

using std::bitset;
using std::string;

//convert_2_10:
//takes a string containing an 8-bit binary number
//and returns the integer decimal equivalent.
//string[0] = high bit, string[7] = low bit
int convert_2_10 (const string bits)
{
	int decimal = 0;

	if(bits[0] == '1')
	{
		decimal = decimal + 128;
	}
	if(bits[1] == '1')
	{
		decimal = decimal + 64;
	}
	if(bits[2] == '1')
	{
		decimal = decimal + 32;
	}
	if(bits[3] == '1')
	{
		decimal = decimal + 16;
	}
	if(bits[4] == '1')
	{
		decimal = decimal + 8;
	}
	if(bits[5] == '1')
	{
		decimal = decimal + 4;
	}
	if(bits[6] == '1')
	{
		decimal = decimal + 2;
	}
	if(bits[7] == '1')
	{
		decimal = decimal + 1;
	}

	return decimal;
}


int main ()
{

  //variables to be used in the processing
  char input_filename[100];
  char output_filename[100];
  char * buffer;
  long size;
```

```
cout << "|~~~~~~~~~~~~~~~~~~~~~~~|" << endl
     << "| WAV Processing Script |" << endl
     << "|-----------------------|" << endl
     << "|                       |" << endl
     << "| Author:  Giles Cotter |" << endl
     << "|                       |" << endl
     << " ~~~~~~~~~~~~~~~~~~~~~~~ " << endl;

//prompt and get data file name from user
cout << "Please enter the WAV data file to process: ";
cin >> input_filename;
cout << endl << "The input file to be used is '" << input_filename << "'." << endl
     << endl;

//prompt and get desired output filename from user
cout << "Please enter the output data file to save to: ";
cin >> output_filename;
cout << endl << "The output file to be used is '" << output_filename << "'." << endl
     << endl;
cout << "Press any key to begin processing..." << endl;
getchar();

//setup input filestream, and read in file into storage buffer
ifstream input(input_filename, ios::in|ios::binary|ios::ate);
size = input.tellg();
input.seekg(0, ios::beg);
buffer = new char [size];
input.read (buffer, size);
input.close();

//report initial data to user
cout << "The complete file is in a buffer." << endl;
cout << "Number of characters collected = " << size << "." << endl;
cout << "Therefore, this file consists of "
     << (size*8) << " bits." << endl << endl
     << "Press any key to continue..." << endl;
getchar();

//***********
//MAIN TASK:  Remove the first 7 bits of every byte of the data,
//             and store the new stream into a new data file.

//One byte in the new file will consist of the lowest order bit of 8 bytes in the
//current data. Therefore, if the current number of bytes is not divisible by 8, you
//will end up with a byte group on the end with 1-7 bits in it.  This cannot be
//outputted correctly.  Therefore, the first thing to be done is discard the end most
//bytes so that the number of bytes is divisible by 8.

//create a int indicating the number of bytes to actually take the lowest order bit
//from.
int low_size = size - size%8;

//report information about the initial and final files to the user
cout << "# of data bytes available: " << size << endl;
cout << "# of bytes used for lowest order bit: " << low_size << " (divisible by 8)"
     << endl;
cout << "Therefore, there will be " << low_size/8 << " bytes after processing." << endl
     << endl;
cout << "Press any key to continue..." << endl;
getchar();

//open up the output stream to output the final bytes
ofstream output(output_filename, ios::out|ios::binary);

//create a variable to track location through buffer
int buf_location = 0;

//loop once for each new byte to be written
for(int i=0; i<(low_size/8); i++)
{
       //create a bitset to hold the new byte
       bitset<8> current;

       //create a variable to track location in the new bitset
       int bit_location = 7;

       //inner loop to put the low bit of each source byte in the new bitset
       for(int curr = buf_location; curr < buf_location+8; curr++)
       {
              //put current source byte into a temp bitset
              bitset<8> temp(buffer[curr]);
```

```
                        current[bit_location] = temp[0];

                        bit_location--;
                }

                buf_location = buf_location+8;

               //Now we want to output the newly created byte to output file.
               //First, convert the current bitset to a char*
               string binary;

                binary = current.to_string();

                //variable to hold integer decimal representation of the digits
                int decimal = convert_2_10(binary);

                //output the value to file
                output << (char)decimal;
        }

        //close up the output stream
        output.close();

        //***********

        cout << "Processing complete." << endl << endl;

        //Clean up
        delete[] buffer;
        return 0;
}
```

Appendix B: Full Listing of Results

Listed here are the results of the data analysis. Each data file was run through the ENT test suite both before and after processing. Therefore, each listing below contains the results for both unprocessed and processed test runs.

Summary of Test Locations:

Test Case	Location Recorded/FM Station used
Base	Recorded with microphone unplugged (control)
Ambient1	Recorded directly in front of computer
Ambient2	Recorded with microphone on sitting computer case
Ambient3	Recorded with microphone up on desk
Ambient4	Recorded with microphone sitting in the middle of the room
Ambient5	Recorded with microphone in door to adjoining room
Radio1	Recorded with radio set to 87.5 FM
Radio2	Recorded with radio set to 89.0 FM
Radio3	Recorded with radio set to 98.4 FM
Radio4	Recorded with radio set to 99.6 FM
Radio5	Recorded with radio set to 102.8 FM

Table of Processed Data Gathered From ENT:

Test Case	Entropy (bits/byte)	Optimal Compression Reduction	Chi Square Distribution %	Arithmetic Mean	Monte Carlo Value for Pi	Serial Correlation Coefficient
BASE	0.0000	100	0.01	0.00	4.000	undefined
AMBIENT 1	0.7976	90	0.01	5.1512	3.997	0.0456
AMBIENT 2	3.1892	60	0.01	171.9962	1.3771	0.4253
AMBIENT 3	0.2005	97	0.01	1.4532	3.9903	0.5797
AMBIENT 4	0.1689	97	0.01	1.1070	3.9945	0.4518
AMBIENT 5	0.3453	95	0.01	3.5291	3.9679	0.6228
RADIO 1	7.9987	0	10.00	127.1946	3.1419	-0.0003
RADIO 2	7.9990	0	90.00	127.4666	3.1385	0.0008
RADIO 3	7.9988	0	50.00	127.8766	3.1298	0.0031
RADIO 4	7.9989	0	50.00	127.5411	3.1301	-0.0003
RADIO 5	7.9989	0	50.00	127.5422	3.1300	-0.0014

Actual Data Collected From ENT:

==

BASE - recorded with microphone disconnected

==

```
------------
Unprocessed:
------------
D:\Thesis>ent base.bin

Entropy = 0.000000 bits per byte.

Optimum compression would reduce the size
of this 1323000 byte file by 100 percent.

Chi square distribution for 1323000 samples is 337365000.00, and
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 128.0000 (127.5 = random)
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is undefined (all values equal!).

----------
Processed:
----------
D:\Thesis>ent base.out

Entropy = 0.000000 bits per byte.

Optimum compression would reduce the size
of this 165375 byte file by 100 percent.

Chi square distribution for 165375 samples is 42170625.00, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 0.0000 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is undefined (all values equal!).
```

======================================

AMBIENT1 - recorded in front of computer

======================================

```
------------
Unprocessed:
------------
D:\Thesis>ent ambient1.bin

Entropy = 0.163669 bits per byte.

Optimum compression would reduce the size
of this 1323000 byte file by 97 percent.

Chi square distribution for 1323000 samples is 323782224.00, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 127.9978 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.551557 (totally uncorrelated = 0.0).



----------
Processed:
----------
D:\Thesis>ent ambient1.out

Entropy = 0.797604 bits per byte.
```

```
Optimum compression would reduce the size
of this 165375 byte file by 90 percent.

Chi square distribution for 165375 samples is 35486283.88, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 5.1512 (127.5 = random).
Monte Carlo value for Pi is 3.997387708 (error 27.24 percent).
Serial correlation coefficient is 0.045589 (totally uncorrelated = 0.0).
```

===

AMBIENT2 - recorded with microphone on computer case

===

```
------------
Unprocessed:
------------
D:\Thesis>ent ambient2.bin

Entropy = 1.702285 bits per byte.

Optimum compression would reduce the size
of this 1323000 byte file by 78 percent.

Chi square distribution for 1323000 samples is 107936314.98, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 127.9871 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.958804 (totally uncorrelated = 0.0).

----------
Processed:
----------
D:\Thesis>ent ambient2.out

Entropy = 3.189242 bits per byte.

Optimum compression would reduce the size
of this 165375 byte file by 60 percent.

Chi square distribution for 165375 samples is 11516424.07, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 171.9962 (127.5 = random).
Monte Carlo value for Pi is 1.377113417 (error 56.17 percent).
Serial correlation coefficient is 0.425320 (totally uncorrelated = 0.0).
```

==

AMBIENT3 - recorded with microphone on desk

==

```
------------
Unprocessed:
------------
D:\Thesis>ent ambient3.bin

Entropy = 0.066425 bits per byte.

Optimum compression would reduce the size
of this 1323000 byte file by 99 percent.

Chi square distribution for 1323000 samples is 333098359.87, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 127.9995 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.765854 (totally uncorrelated = 0.0).
```

```
----------
Processed:
----------
D:\Thesis>ent ambient3.out

Entropy = 0.200500 bits per byte.

Optimum compression would reduce the size
of this 165375 byte file by 97 percent.

Chi square distribution for 165375 samples is 40955993.91, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 1.4532 (127.5 = random).
Monte Carlo value for Pi is 3.990276468 (error 27.01 percent).
Serial correlation coefficient is 0.579729 (totally uncorrelated = 0.0).
```

```
==================================================
```

AMBIENT4 - recorded with microphone in middle of room

```
==================================================

------------
Unprocessed:
------------
D:\Thesis>ent ambient4.bin

Entropy = 0.052910 bits per byte.

Optimum compression would reduce the size
of this 1323000 byte file by 99 percent.

Chi square distribution for 1323000 samples is 334054960.48, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 127.9997 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.766098 (totally uncorrelated = 0.0).

----------
Processed:
----------
D:\Thesis>ent ambient4.out

Entropy = 0.168896 bits per byte.

Optimum compression would reduce the size
of this 165375 byte file by 97 percent.

Chi square distribution for 165375 samples is 41171656.75, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 1.1070 (127.5 = random).
Monte Carlo value for Pi is 3.994485161 (error 27.15 percent).
Serial correlation coefficient is 0.451847 (totally uncorrelated = 0.0).
```

```
=========================================================
```

AMBIENT5 - recorded with microphone in door to adjoining room

```
=========================================================

------------
Unprocessed:
------------
D:\Thesis>ent ambient5.bin

Entropy = 0.172124 bits per byte.

Optimum compression would reduce the size
of this 1323000 byte file by 97 percent.

Chi square distribution for 1323000 samples is 326161051.50, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 127.9992 (127.5 = random).
```

```
Monte Carlo value for Pi is 3.999655329 (error 27.31 percent).
Serial correlation coefficient is 0.978328 (totally uncorrelated = 0.0).


----------
Processed:
----------
D:\Thesis>ent ambient5.out

Entropy = 0.345328 bits per byte.

Optimum compression would reduce the size
of this 165375 byte file by 95 percent.

Chi square distribution for 165375 samples is 39945095.92, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 3.5291 (127.5 = random).
Monte Carlo value for Pi is 3.967926856 (error 26.30 percent).
Serial correlation coefficient is 0.622835 (totally uncorrelated = 0.0).
```

================

RADIO1 - 87.5 FM

================

```
------------
Unprocessed:
------------
D:\Thesis>ent radio1.bin

Entropy = 4.349350 bits per byte.

Optimum compression would reduce the size
of this 1323000 byte file by 45 percent.

Chi square distribution for 1323000 samples is 17856902.83, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 127.9904 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.892238 (totally uncorrelated = 0.0).

----------
Processed:
----------
D:\Thesis>ent radio1.out

Entropy = 7.998728 bits per byte.

Optimum compression would reduce the size
of this 165375 byte file by 0 percent.

Chi square distribution for 165375 samples is 291.40, and randomly
would exceed this value 10.00 percent of the times.

Arithmetic mean value of data bytes is 127.1946 (127.5 = random).
Monte Carlo value for Pi is 3.142732748 (error 0.04 percent).
Serial correlation coefficient is -0.000279 (totally uncorrelated = 0.0).
```

================

RADIO2 - 89.0 FM

================

```
------------
Unprocessed:
------------
D:\Thesis>ent radio2.bin

Entropy = 5.137604 bits per byte.

Optimum compression would reduce the size
of this 1323000 byte file by 35 percent.
```

```
Chi square distribution for 1323000 samples is 10221666.30, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 127.9906 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.902722 (totally uncorrelated = 0.0).

----------
Processed:
----------
D:\Thesis>ent radio2.out

Entropy = 7.999044 bits per byte.

Optimum compression would reduce the size
of this 165375 byte file by 0 percent.

Chi square distribution for 165375 samples is 219.60, and randomly
would exceed this value 90.00 percent of the times.

Arithmetic mean value of data bytes is 127.4666 (127.5 = random).
Monte Carlo value for Pi is 3.138524055 (error 0.10 percent).
Serial correlation coefficient is 0.000822 (totally uncorrelated = 0.0).
```

================

RADIO3 - 98.4 FM

================

```
------------
Unprocessed:
------------
D:\Thesis>ent radio3.bin

Entropy = 4.291017 bits per byte.

Optimum compression would reduce the size
of this 1323000 byte file by 46 percent.

Chi square distribution for 1323000 samples is 18884650.82, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 127.9906 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.883871 (totally uncorrelated = 0.0).

----------
Processed:
----------
D:\Thesis>ent radio3.out

Entropy = 7.998849 bits per byte.

Optimum compression would reduce the size
of this 165375 byte file by 0 percent.

Chi square distribution for 165375 samples is 264.96, and randomly
would exceed this value 50.00 percent of the times.

Arithmetic mean value of data bytes is 127.8766 (127.5 = random).
Monte Carlo value for Pi is 3.129816414 (error 0.37 percent).
Serial correlation coefficient is 0.003124 (totally uncorrelated = 0.0).
```

================

RADIO4 - 99.6 FM

================

```
------------
Unprocessed:
------------
D:\Thesis>ent radio4.bin

Entropy = 4.319992 bits per byte.
```

```
Optimum compression would reduce the size
of this 1323000 byte file by 46 percent.

Chi square distribution for 1323000 samples is 18523807.87, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 127.9910 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.886075 (totally uncorrelated = 0.0).

----------
Processed:
----------
D:\Thesis>ent radio4.out

Entropy = 7.998922 bits per byte.

Optimum compression would reduce the size
of this 165375 byte file by 0 percent.

Chi square distribution for 165375 samples is 247.66, and randomly
would exceed this value 50.00 percent of the times.

Arithmetic mean value of data bytes is 127.5411 (127.5 = random).
Monte Carlo value for Pi is 3.130106669 (error 0.37 percent).
Serial correlation coefficient is -0.000295 (totally uncorrelated = 0.0).
```

=================

RADIO5 - 102.8 FM

=================

```
------------
Unprocessed:
------------
D:\Thesis>ent radio5.bin

Entropy = 4.844362 bits per byte.

Optimum compression would reduce the size
of this 1323000 byte file by 39 percent.

Chi square distribution for 1323000 samples is 12666211.53, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 127.9908 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is 0.893247 (totally uncorrelated = 0.0).

----------
Processed:
----------
D:\Thesis>ent radio5.out

Entropy = 7.998914 bits per byte.

Optimum compression would reduce the size
of this 165375 byte file by 0 percent.

Chi square distribution for 165375 samples is 248.59, and randomly
would exceed this value 50.00 percent of the times.

Arithmetic mean value of data bytes is 127.5422 (127.5 = random).
Monte Carlo value for Pi is 3.129961541 (error 0.37 percent).
Serial correlation coefficient is -0.001425 (totally uncorrelated = 0.0).
```

Appendix C: WAV File Format

This the standard format for a WAV file [7].

RIFF Chunk (12 bytes in length total)

Byte Number	
0 - 3	"RIFF" (ASCII Characters)
4 - 7	Total Length Of Package To Follow (Binary, little endian)
8 - 11	"WAVE" (ASCII Characters)

FORMAT Chunk (24 bytes in length total)

Byte Number	
0 - 3	"fmt_" (ASCII Characters)
4 - 7	Length Of FORMAT Chunk (Binary, always 0x10)
8 - 9	Always 0x01
10 - 11	Channel Numbers (Always 0x01=Mono, 0x02=Stereo)
12 - 15	Sample Rate (Binary, in Hz)
16 - 19	Bytes Per Second
20 - 21	Bytes Per Sample: 1=8 bit Mono, 2=8 bit Stereo or 16 bit Mono, 4=16 bit Stereo
22 - 23	Bits Per Sample

DATA Chunk

Byte Number	
0 - 3	"data" (ASCII Characters)
4 - 7	Length Of Data To Follow
8 - end	Data (Samples)