Location Dependent Cryptosystem

Kunal Mukherjee, Computer Engineering

Sponsored by Mr. Mike Ciholas

Academic Advisor: Dr. Donald Roberts

Design Advisor: Mr. Justin Bennett

Software Advisor: Mr. Tim DeBaillie

April 21, 2019

Evansville, Indiana

Acknowledgements

I would like to thank my sponsor, Mr. Mike Ciholas, and my academic advisor, Dr. Donald Roberts. Mr. Ciholas, who is the owner of Ciholas, Inc. which is located in Newburgh, IN, sponsored me as well as provided me with lab equipment and Ciholas' devices for this project free of cost. Dr. Roberts, who is an Associate Professor of Computer Science at the University of Evansville met with me every two weeks and gave me helpful suggestions on how to implement and design my application that will make it helpful to build upon later.

I am also thankful for my design and software mentors, Mr. Justin Bennett and Mr. Tim DeBaillie, respectively. Despite their demanding schedules, they carefully critiqued my concepts and helped me prepare for the nuisances that arrive when implementing a software concept using hardware upon a pre-built system. Both individuals gave me detailed feedback on my technical writing, as well as helped me improve my presentation skills so that my work is clear to any audience.

Table of Contents

List of Figures

I. Introduction

The 21st century has brought many important inventions and innovations to academics as well as research driven industries. While some innovations are valuable, some are malicious such as digital piracy and intellectual property theft. Certain research dependent industries need new security measures to protect their intellectual property from corporate or international espionage. Usually, the intellectual property theft happens during data transfer. The entertainment industry has tried counteracting this problem by encrypting movies before sending them to movie theaters with the impression that only authorized employees with the key or password can decrypt the movies. They overlook a vulnerability in the solution that the key of the encrypted movie needs to be transferred from the movie producers to the respective theaters either electronically or physically. This could potentially be intercepted by the unintended user, and they could then trivially decrypt the encrypted movie file. Therefore, a viable solution is that the user should only be able to decrypt a file if he or she is at an acceptable geographical location that is pre-authorized by the sender. Therefore, this project is going to create a system that will transfer the generated key in such a way that only the user at the correct location will be able to decipher it and then decrypt the file without the receiver's assistance or knowledge of the password. Therefore, the need for sharing the password electronically or physically, is omitted. To develop this solution and strengthen the security against piracy for the entertainment industry as well as research-driven fields, I am researching and developing a location dependent cryptosystem. The cryptosystem will allow the user to decrypt an encrypted data, only if the user is at an approved location which is predetermined by the sender.

II. Background

The current encryption standards for industries are AES256 (128 bits or higher), TDES (double-length keys), ECC (160 bits or higher) and ElGamal (1024 bits or higher) [1]. These encryption standards are secure as the underlying base problem, the discrete log problem (DLP), is intractable and exponentially hard for large primes [2]. Intractable is defined as taking thousands of years to brute force through the function, even for the top five supercomputers of the world [2]. In industry, the encryption standard used to encrypt data is AES [3]. The encrypted data is then transmitted using any one of the cryptographic network protocols, such as Internet Protocol Security (IPsec) [4]. However, the key of the encryption is independent of the location and it needs to be transferred to the receiver physically or electronically. Therefore, to make sure the encrypted data can only be decrypted at the authorized location, the key of the encrypted data is associated with the approved location. The system will start the decryption process automatically without the need for the receiver's assistance, thus protecting the password's integrity.

The crypto system will have the following properties and accomplish the following responsibilities:

- The password of the encrypted data will not be transmitted in the user or machine decipherable form
    - The key will be transmitted in such a way that it cannot be deciphered unless the receiver is in the authorized location
- There will be an authorized zone, rather than a specific point in space so that the user has a certain degree of tolerance with his or her location

- Anyone in the authorized zone is an authorized user and can generate the key just by receiving the encrypted data packets
- Any unauthorized user will not be able to generate the key
- User assistance is not needed for the decryption process to start

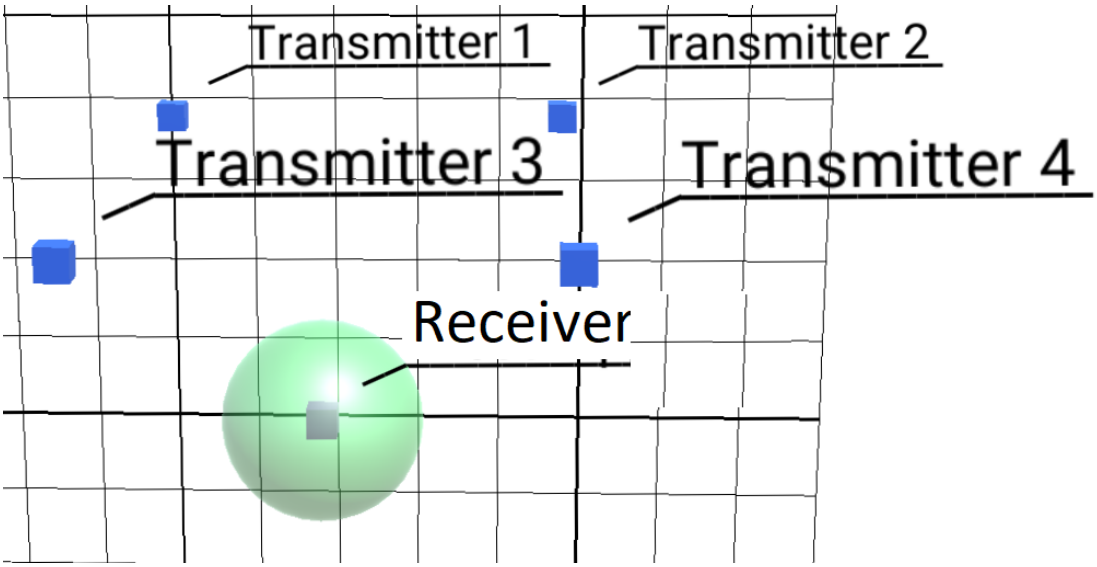

Figure 1: Cryptosystem

III. Project Design

A. Hardware

This project required a device that can emit ultra-precise timing information. Ciholas' server NetApp can relay DWETH101 timing information accurately up to eight nanoseconds. Therefore, I chose Ciholas DWETH101. This hardware contains a DecaWave chip that can emit timestamp accurately up to ten picoseconds. However, the Ciholas server uses the DecaWave timestamp and makes it more precise by using the most

significant bits to get information in the nanoseconds range. In terms of distances, the devices are accurately up to ±3 cm. A DWETH101 is shown in Figure 2.

Figure 2: DWETH101 Module

B. Software

i. Audio Extraction

The software was developed in multiple parts according to the final project requirements. The final demonstration of the project involved audio signal. Therefore, the development of an audio extraction tool was important. The audio extraction script takes an audio file, “input.wav,” and converts it into principal frequency values. The values are then stored in a file, which can later be used to convert the text file back into audio. The code is provided in Appendix A. The script

uses the *ffmpeg* utility to convert the .wav file to principal frequency values and vice-versa.

ii. AES-256 and SHA-256 Implementation

The project is based on AES-256 encryption. The encryption process takes a specific buffer, e.g. 32 bytes of characters. The AES-256 encrypts the buffer using an initialization vector or IV and the key provided. The AES encryption creates 32 bytes or 1 block of encrypted data from the unencrypted buffer. The encrypted block needs to be stored in a text file. This encryption process continues until all data has been converted into encrypted block.

The decryption process takes the encrypted data as well as the password and recreates the unencrypted data. For the cryptosystem, the password provided by the user is converted to fixed 32 bytes of character using SHA256. A SHA256 hashing function has two important properties. First, it protects the integrity of the user's password since no one has knowledge of the plain text key except the user. Secondly, it creates a fixed length key every time, thus broadening the collision domain of the encrypted passwords. The code is provided in Appendix B.

iii. JMTK Protocol

The JMTK or Gem Tech protocol is the methodology that takes the 32 bytes of the SHA password and transmits it using the difference of timestamps. 32 bytes of SHA to transmit 33 packet transmission is needed as the first transmission is used as reference for the first SHA calculation.

Timestamp is the time the Ciholas server, NetApp, transmits when one of the devices, DWETH101, connected to it receives a packet or transmits a packet. If a device receives a packet, it is called reception timestamp, Trx. Additionally, if the device transmits a packet, it is called transmission timestamp, Ttx. The Ciholas network server runs an internal server clock which gives time in Network Time ticks or NT ticks. K_f is a conversion factor that converts second to NT ticks and is numerically equal to 975000 * 65536 or 6,389,760,000. One NT tick is equal to 16.5 nanoseconds or one second is 6,389,760,000 NT ticks. This shows how precise and accurate the Ciholas server is.

The JMTK protocol had to be made compatible with the NetApp. If the protocol asked the NetApp to transmit one of its packets when the NetApp was scheduled to transmit one of NetApp's packet, then the JMTK packet would not be transmitted due to NetApp's internal packet taking higher priority. Thus, an initial offset had to be introduced for the protocol to work. This ensured the protocol did not interfere with the scheduling of the NetApp's internal packets and also marked the beginning of the 33-timestamp transmission sequence. The initial offset is called $T_{startOffset}$ and is numerically equal to 5 milliseconds or 319488000 (NT ticks).

Time window is the amount of time after which the NetApp sends out its internal packets to make sure that all devices connected to it are time synchronized. This occurs after 100 milliseconds. Therefore, another offset, T_{net} or the network offset, is added so that the 33 transmissions can happen in one-time window. The network offset is numerically equal to n * K_f NT ticks, where n represents a value that can be found by taking current networking of the NetApp and dividing it by

K_f. Hence, to begin a transmission and to give the reference for the first SHA calculation, a packet is transmitted at T_{net} plus $T_{startOffset}$. T_{net} plus $T_{startOffset}$ is called initial network cadence. A transmission time window can be seen using Figure 3 below.

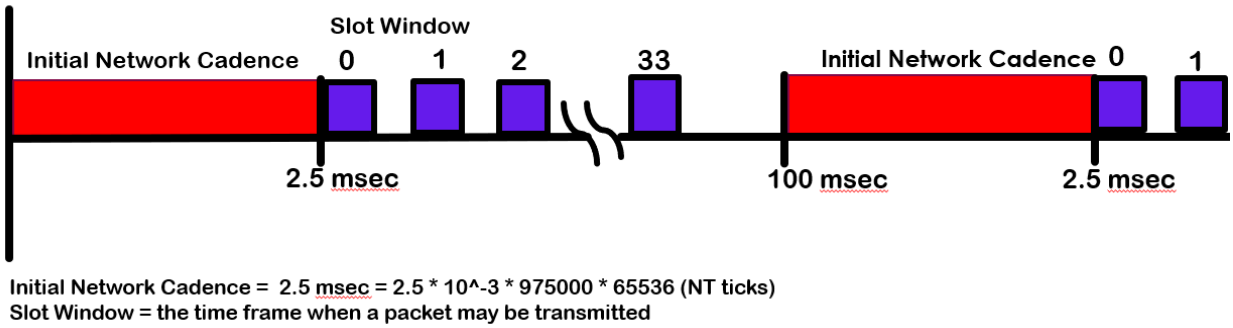

Figure 3: Time Window Diagram

Once the protocol has determined which time window it is transmitting as well as the first transmission packet, the protocol then schedules the next 32 packet transmission so the difference can give the SHA back. The 32 packet transmission has to have at least a 2.5 millisecond difference between the transmissions. This is the time the devices need to process the packet received and to again go back to listening mode. This time difference is called $T_{betweenOffset}$. $T_{betweenOffset}$ is known by both the receiver and the transmitter side. The transmitter also knows the time of flight a packet takes from each anchor to the approved location, T_{distA}.

Next, the protocol looks at the amount of time allocated per slot corresponding to the SHA values, T_{slot}. T_{slot} depends on the amount of approved space of the decryption region. For example, a decryption region, sphere of radius 2 meters would be 2 meters divided by C or 6.667 nanoseconds. There are 256 time

slots corresponding to the 255 values (0 to FF) the SHA can take. A transmission slot window can be seen in Figure 4 below.

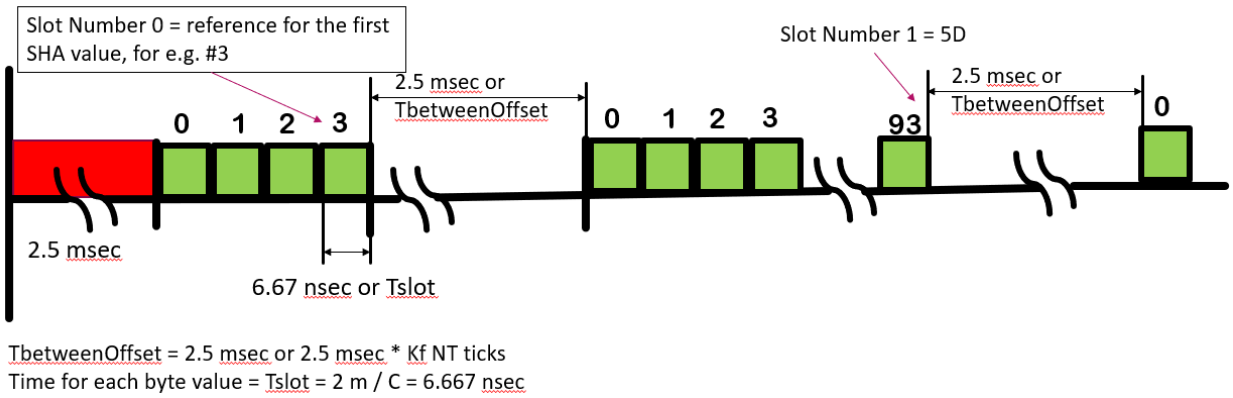

Figure 4: Slot Window

After determining all of the constants, the JMTK protocol looks at which values of the SHA need to be transferred. 5D will be used as an example. The transmission side will take the last transmission timestamp calculated, $T_{tx}(n-1)$, and add the time of flight of the packet from the last chosen anchor to the reception region, $T_{distA}(n-1)$. This gives the transmitter the reference point. Then, the transmission side adds $T_{betweenOffset.}$ This ensures that the receiver is in the listening state. Then, the transmission side takes the SHA value SlotNumber 5D or 93 and multiplies it by T_{slot}. The final step is to subtract the time of flight, $T_{tx}(n)$, to account for the time it takes from this current packet to go from the current transmitter to the approved location. Therefore, the transmission equation is, $T_{tx}(n) = T_{tx}(n-1) + T_{distA}(n-1) + T_{betweenOffset} + (SlotNumber * T_{slot}) - T_{distA}(n)$.

After the receiver accepts a second timestamp, the receiver first saves this timestamp as it becomes the reference for the second SHA transfer. Secondly, the receiver utilizes the second timestamp to get the first SHA value. The reception

side takes the second timestamp and subtracts the first timestamp from it. Then, the reception side subtracts $T_{betweenOffset}$ from it. As a result, the receiver is now left with T_{slot} times the SHA value. Therefore, after dividing the result by $T_{slot,}$ the slot value is equal to the first SHA transmitted value. Thus, the reception equation is the following: SHA value(n) = (T_{rx}(n) - T_{rx}(n-1) - $T_{betweenOffset}$) / T_{slot}. Figure 5 shows how the different constant parameters can be visualized in a reception timeline.

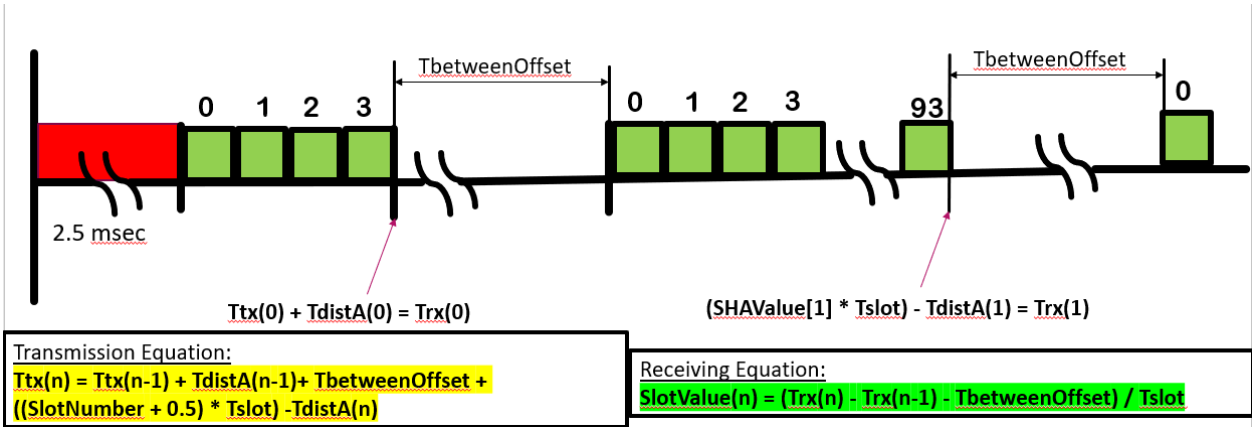

Figure 5: Transmission as well as reception equation quantities diagram on reception timeline

The eavesdropper is a receiver who is not at the approved location, represented in Figure 8. For the eavesdropper to get the SHA byte, he or she must overcome two safety parameters. First, the eavesdropper must know the reference timestamp, T_{rx}(n-1), from which the second timestamp would be subtracted. Secondly, the eavesdropper must know the distances, T_{distA}(n-1) and T_{distA}(n), from which both packet transmissions are made. Without this information, the resultant of the second timestamp minus the first timestamp cannot be divided by the T_{slot} value to get the correct SHA value. From using Figure 6 and 7, one can

observe that being just 1.06 meters away from the approved location can change the SHA value.

Step 1: Emit the first packet or the starting time stamp. The receiver will get a bogus slot number, which lets the receiver know transmission is going to begin

Transmission Side

Assume: Anchor T1 is selected and it is 1 m from the approved location. Therefore, Tslot is 383 NT ticks.

Ttx(0) = Tnet + TstartOffset

= 6389760000 + 319488000 = **6709248000**

Approved Receiver

Trx(n-1) = (Trx(n) - Trx(n-1) - TbetweenOffset) / Tslot

= (**6709248000** – 0 – 159744000) / (426) = **15374423**

Evesdropper

Trx(n-1) = (Trx(n) - Trx(n-1) - TbetweenOffset) / Tslot

= (**6708279048** – 0 – 159744000) / (426) = **15372148**

Figure 6: Example calculation showing the difference between the reference value calculated by the intended receiver and eavesdropper

Step 2: Emit the 1st byte of a hash (E.g. HEX: 6D or DEC: 109)

Transmission Side

Assume: Anchor T2 is selected and it is 2 m from the approved location. Therefore, Tslot is **667** NT ticks

Ttx(n) = Ttx(n) = Ttx(n-1) + TdistA(n-1)+ TbetweenOffset- **TdistA(n)**+ ((**SlotNumber** + 0.5) * Tslot)

= 15374423 + 383 + 159744000 – **667** + ((**109** + 0.5) * 426) = **175164857**

Approved Receiver

Trx(n) = (**Trx(n)** - Trx(n-1) - TbetweenOffset) / Tslot

SlotValue(0) = (**175164857** – 15374423 – 159744000) / (426) = **109** or **6D**

Eavesdropper

SlotValue(0) = (**175162582** – 15372148 – 159744000) / (426) = **107** or **6B**

175164857 – 175162582 = 2275 NT ticks = 35.6 nsec ≈ 1.06 meters

Figure 7: Example calculation showing the difference between the SHA calculated by the intended receiver and eavesdropper

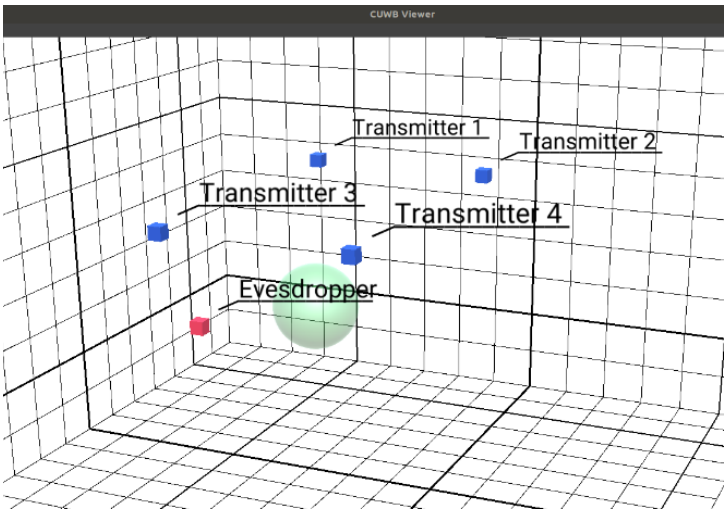

Figure 8: The eavesdropper and the crypto system

iv. Cryptosystem Simulator

The cryptosystem simulator was developed as proof of concept to see if the transmission and the reception equation will work or not. The cryptosystem was built using TCP, which substituted UWB. Because the real-world system will have noise in the air, the simulation changed the reception equation by adding T_{noise} to account for the noise in the atmosphere. As a result, the reception equation became SHA value(n) = $(T_{rx}(n) - T_{rx}(n-1) - T_{betweenOffset} + T_{noise}) / T_{slot}$.

The simulator server took an audio file, password, and the distances of the three anchors from the approved location. The server first created a SHA out of the password and encrypted the audio file. Then, the server divided the encrypted data into 800 bytes and sent them as payload with every packet transfer. The server also sent the transmission timestamp as payload. The transfer terminated when all of the

encrypted data had been transferred. The first packet for the transfer T_{net} was chosen to be equal to n * K_f NT ticks.

The simulator client connected, and the transfer began. The client utilized the transmission timestamps to generate the 32-byte SHA value. The client then used the SHA value to decrypt the received encrypted data. After the decryption process was over, the client played the decrypted data back. If the correct SHA had been transmitted, it would play the same audio back as intended by the server. If not, the client will just play noise, which in turn means that the decryption process was unsuccessful.

The cryptosystem simulator worked with the atmospheric noise involved and showed that with UWB as the medium of transfer, the cryptosystem would function. Cryptosystem simulator code is provided in appendix C.

v. Cryptosystem Server and Client

The cryptosystem server listens to the NetApp's timing packets and keeps track of the time window so that this server can transmit the 33 packets in one-time window. First, a C script takes the password from the sender and encrypts the data using the SHA of the password. Then, the C script starts a python script which selects an appropriate T_{net} to use as the first packet transmission time and transmits the first packet with a payload of 800 bytes of encrypted data. This process keeps on going until all of the encrypted data has been transferred.

The approved receivers keep getting the packets and use the received packets' timestamps to create the SHA back. The receiver waits for 2 seconds. If the receiver receives no packet within the 2 seconds, it stops listening and starts a

C script that takes the SHA generated and uses it to decrypt the data received. After the decryption process is over, the C script presents the data to the user, e.g. plays the audio file back to the user. The cryptosystem server and client's both C and python code are provided in Appendix D.

C. Social, Political and Environmental Impact

There are no social or environmental concerns associated with this project. This is a research endeavor to create a secure and robust cryptosystem. This will benefit everyone as well as improve the social aspect of sharing. This project will make sharing secure and trustworthy. Ciholas hardware is FCC approved [5]. As a result, it does not possess any environmental or health risks since the electromagnetic radiation is under the approved limit.

There is a political concern with this project. If this project becomes completely robust without any vulnerabilities, then different research as well as secret agencies would like to receive access to the project and try to stop this information from spreading to other parties. Therefore, I will keep an online updated version of my project available, which can be accessible from anywhere. Thus, if anything happens to me or the people associated with this project, the knowledge as well as the mechanism of the cryptosystem would not be lost. Therefore, no secret agency will have advantage over the other.

D. Industry Standards, Health and Safety Consideration

There are no comparable standards that map the password of an encryption to location and transmit the password wirelessly to the receiver. However, the encryption

protocol used to encrypt the user data before transmission will be done using the industry standard encryption protocol for data transfer AES-256. Therefore, AES-256 guarantees that even if the packet is intercepted by an unintended receiver during transmission, the receiver will not be able to decipher the packets.

IV. Results

The cryptosystem was able to demonstrate all of the requirements and features as stated during the developmental process. The cryptosystem was able to transfer the SHA of the password in a form that was neither machine nor human readable. The approved location was a space rather than a point. This ensured that the user had a certain degree of freedom and the location had a certain tolerance. The cryptosystem also started the decryption process without the receiver's assistance and any receiver not at the approved location did not receive the correct SHA value.

V. Future Work

Even with these security features, this system can be defeated by implementing many secondary anchors and using them to triangulate the approved location. The secondary anchors are used to determine which anchor sent which packets and their relative times of flight. The process is not easy, but feasible. Thus, it cannot be readily deployed at military and entertainment facilities.

Moreover, an important feature can be developed called a rolling key encryption-system. With this, the receiver will have to immediately transmit a specific packet when

he or she receives an encrypted packet. The server will determine if the received timestamp of this specific packet is from the approved location or not. If not, the server will change the encryption key and re-transmit. If so, the next packet will be sent. This will continue until all of the packets have been transmitted and the cryptosystem can function normally.

Appendix A. Audio Extraction Code

```
//Programmer: Kunal Mukherjee                        # Date completed:

// clang Audio_extraction.c -o driver

//adding the header files
#include <stdio.h>
#include <math.h>
#include <stdint.h>


int main(int argc, char const *argv[])
{

    //open the respective files // 44.1 * 15100
    FILE * inputAudiofile;
    inputAudiofile  = popen("ffmpeg -i input.wav -hide_banner -f s16le -ac 1
-", "r");

    FILE * tempAudiosampleFile;
    tempAudiosampleFile = fopen("tempAud.txt","w");

    // Read, modify and write one sample at a time
    int16_t audioSample;

    while( fread(&audioSample, 2, 1, inputAudiofile)) // read one 2-byte
sample
    {
        fprintf(tempAudiosampleFile, "%hd ", audioSample);
    }

    pclose(inputAudiofile);
    fclose(tempAudiosampleFile);

    //opening a temp file to put the values in
    FILE * outputAudiofile;
    outputAudiofile = popen("ffmpeg -y -f s16le -ar 44100 -ac 1 -i -
output.wav -hide_banner", "w");

    FILE * tempAudiosampleFileInput;
    tempAudiosampleFileInput = fopen("tempAud.txt","r");

    int16_t audioSampleRead;

    while(fscanf(tempAudiosampleFileInput, "%hd" , &audioSampleRead) != EOF)
    {
        fwrite(&audioSampleRead, 2, 1, outputAudiofile);
    }

    // Close input and output pipes
    pclose(outputAudiofile);
    fclose(tempAudiosampleFileInput);

    FILE * playResultAudio;
    playResultAudio = popen("ffplay -hide_banner -autoexit output.wav", "r");
```

```
    pclose (playResultAudio);

    return 0;
}
```

Appendix B. AES256 Implementation Code

```
// File: Makefile
// Class: EE 495-Senior Reseach            # Instructor: Mr. Mike Ciholas and
Dr. Donald Roberts
// Assignment: Location-Dept Cryptosystem  # Date started: 12/26/2018
//Programmer: Kunal Mukherjee              # Date completed:

//adding the header files
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <mcrypt.h> //http://linux.die.net/man/3/mcrypt

//defining the max charc size
#define MAX_CHARACTER_SIZE 32

//the encrypt function
int encrypt(
    void* buffer,
    int buffer_len, /* Because the plaintext could include null bytes*/
    char* IV, //initilization vector
    char* key,
    int key_len
);

//the decrypt function
int decrypt(
    void* buffer,
    int buffer_len,
    char* IV,
    char* key,
    int key_len
);

//cipher text displyer
void printEncryptedFile(char* ciphertext,
                        int len,
                        FILE* encyptFile);

int main(int argc, char const *argv[])
{
          //check to see if all the argv is entred
    if (argc != 2)
    {
      printf("Usage: ./server <password> \n");
      return 0;
    }

    //Step 1: Take Audio file -> Digitalize the signal
    printf("==Location-Dependent Algorithm==\n\n");

    //open the respective audio and audio_text input files
    FILE * inputAudiofile;
```

```
    inputAudiofile  = popen("ffmpeg -i input.wav -hide_banner -f s16le -ac 1
-", "r");

    FILE * tempAudiosampleFile;
    tempAudiosampleFile = fopen("tempAud.txt","w");

    // Read, modify and write one sample at a time
    int16_t audioSample;

    while(fread(&audioSample, 2, 1, inputAudiofile)) // read one 2-byte
sample
    {
        fprintf(tempAudiosampleFile, "%hd ", audioSample);
    }

    //closing the pipe and file
    pclose(inputAudiofile);
    fclose(tempAudiosampleFile);

    printf("\nAudio file extractd and audio sample file created.\n");

    //Step 2: Take the digitalized signal and encode with AES
    printf("\nEncryption Process Started\n");

    //open the audio_text input and the encrypted file
    FILE * inputFile;
    inputFile = fopen("tempAud.txt","r");

    FILE * encyptFileOutput;
    encyptFileOutput = fopen("EncrFileOutput.txt","w");

    //create a MCRYPT to get certain info
    MCRYPT td = mcrypt_module_open("rijndael-256", NULL, "cbc", NULL);

          //A random block should be placed as the first block (IV)
          //so the same block or messages always encrypt to something
different.
          char * IVEncr = malloc(mcrypt_enc_get_iv_size(td)); //return 8
          FILE * fp;
    fp = fopen("/dev/urandom", "r");
    fread(IVEncr, 1, mcrypt_enc_get_iv_size(td), fp);
    fclose(fp);
    mcrypt_generic_end(td);
    //place the IV in the encrypted file
    printEncryptedFile(IVEncr , mcrypt_enc_get_iv_size(td) ,
encyptFileOutput);

    //check to see if the key is MAX_CHARACTER_SIZE charcter long
    char * keyEncr = calloc(1, MAX_CHARACTER_SIZE); //MAX_CHARACTER_SIZE * 8
= 128
    strncpy(keyEncr, argv[1], MAX_CHARACTER_SIZE);
          int keyEncrsize = MAX_CHARACTER_SIZE; /* 256 bits */

    //initialize the buffer
          int bufferEncr_len = MAX_CHARACTER_SIZE;
    char * bufferEncr = calloc(1, bufferEncr_len);
```

```
    //Encryption algorithm info display
    printf("The IV is: %s\n", IVEncr);
    printf("The Key is: %s\n", keyEncr);

    while(fgets(bufferEncr, sizeof bufferEncr, inputFile) != NULL)
    {
      //process buffer
      encrypt(bufferEncr, bufferEncr_len, IVEncr, keyEncr, keyEncrsize);
      printEncryptedFile(bufferEncr , bufferEncr_len , encyptFileOutput);
    }
    if (feof(inputFile))
    {
      // hit end of file
      printf("Encryption Process Completed\n\n");
    }

    //closing file and free memory after encryption
    fclose(inputFile);
    fclose(encyptFileOutput);

    free(IVEncr);
    free(bufferEncr);
    free(keyEncr);

    //Step3: Take the AES encoding and convert it back to digitalized signal
    //decryption algorithm
    printf("Decryption Process Started\n");

    //open the encrypted file to get the IV and the AES blocks
    //open the output_digitalized text that would be created
    FILE * encyptFileInput;
    encyptFileInput = fopen("EncrFileOutput.txt","r");

    FILE * outputFile;
    outputFile = fopen("DecrOutput.txt","w");

    //initialize decrypt buffer
    int bufferDecr_len = MAX_CHARACTER_SIZE;
    char * bufferDecr = calloc(1, bufferDecr_len);

    //initialize the key
    char * keyDecr = calloc(1, MAX_CHARACTER_SIZE); //MAX_CHARACTER_SIZE * 8
= 128
    strncpy(keyDecr, argv[1], MAX_CHARACTER_SIZE);
    int keyDecrsize = MAX_CHARACTER_SIZE;

    //getting IV
    char * IVDecr = calloc(1, MAX_CHARACTER_SIZE);
    int IVBuf = 0;

    for (int i = 0; i < MAX_CHARACTER_SIZE; i++)
    {
      fscanf(encyptFileInput, "%d" , &IVBuf);
      IVDecr[i] = IVBuf;
    }

    //display the decrypting info
```

```
    printf("The IV is: %s\n", IVDecr);
    printf("The Key is: %s\n", keyDecr);

    //initilizing he tempAESbuffer and bufferIndex
    int  AESbuf = 0;
    int bufIndex = 0;

    //getting the AES blocks from the encrypt_file and decrypting it
    while(fscanf(encyptFileInput, "%d" , &AESbuf) != EOF)
    {
      if(bufIndex < MAX_CHARACTER_SIZE)
      {
        bufferDecr[bufIndex] = AESbuf;
        bufIndex++;
      }
      else
      {
        decrypt(bufferDecr, bufferDecr_len, IVDecr, keyDecr, keyDecrsize);
        fprintf(outputFile, "%s", bufferDecr);

        bufIndex = 0;
        memset(bufferDecr, 0 , MAX_CHARACTER_SIZE);
        bufferDecr[bufIndex] = AESbuf;
        bufIndex++;
      }
    }

    if( strlen(bufferDecr) != 0)
    {
      decrypt(bufferDecr, bufferDecr_len, IVDecr, keyDecr, keyDecrsize);
      fprintf(outputFile, "%s", bufferDecr);
    }

    printf("Decryption Process Completed\n");

    //closing the output file and free memory
    fclose(encyptFileInput);
    fclose(outputFile);

    free(keyDecr);
    free(bufferDecr);
    free(IVDecr);

    //Step 4: Convert the decrypted file to Audio
    printf("\nConvert the decrypted audio sample to audio file\n");

    //opening a output_digitalized file to create the audio
    FILE * outputAudiofile;
    outputAudiofile = popen("ffmpeg -y -f s16le -ar 44100 -ac 1 -i -
output.wav -hide_banner", "w");

    FILE * tempAudiosampleFileInput;
    tempAudiosampleFileInput = fopen("DecrOutput.txt","r");

    int16_t audioSampleRead;

    while(fscanf(tempAudiosampleFileInput, "%hd" , &audioSampleRead) != EOF)
```

```
    {
        fwrite(&audioSampleRead, 2, 1, outputAudiofile);
    }

    // Close input and output pipes
    pclose(outputAudiofile);
    fclose(tempAudiosampleFileInput);

    //process completion display
    printf("\n*** Location-Dependent Algorithm Process Completed ***\n\n");

    //play the resultent audio
    FILE * playResultAudio;
    playResultAudio = popen("ffplay -hide_banner -autoexit output.wav", "r");
    pclose (playResultAudio);

    return 0;
}

int encrypt(void* buffer, int buffer_len, char* IV, char* key, int key_len)
{
  MCRYPT td = mcrypt_module_open("rijndael-256", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  if( buffer_len % blocksize != 0 )
    {
      return 1;
    }

  mcrypt_generic_init(td, key, key_len, IV);
  mcrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
  mcrypt_module_close(td);

  return 0;
}

int decrypt(void* buffer, int buffer_len, char* IV, char* key, int key_len)
{
  MCRYPT td = mcrypt_module_open("rijndael-256", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  if( buffer_len % blocksize != 0 )
    {
      return 1;
    }

  mcrypt_generic_init(td, key, key_len, IV);
  mdecrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
  mcrypt_module_close(td);

  return 0;
}

//displays as well as writes the encrypt file
void printEncryptedFile(char* ciphertext, int len, FILE* encyptFile)
{
  int v;
```

```
  for (v=0; v<len; v++)
  {
    fprintf(encyptFile, "%d", ciphertext[v]);
    fprintf(encyptFile, "%s", " ");
  }

  fprintf(encyptFile, "%s", "\n");
}
```

Appendix C. Cryptosystem Simulator Code

i. *TCP Server*

```
// File: Makefile
// Class: EE 495-Senior Reseach            # Instructor: Mr. Mike Ciholas and
Dr. Donald Roberts
// Assignment: Location-Dept Cryptosystem  # Date started: 3/5/2018
//Programmer: Kunal Mukherjee              # Date completed:

//adding the header files for encryption
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <mcrypt.h> //http://linux.die.net/man/3/mcrypt
//sudo apt-get install libmcrypt-dev
//sudo apt-get install ffmpeg

//header for Tx values
//sudo apt-get install libssl-dev
#include <openssl/sha.h>
#include <time.h>

//header for TCP tranfer
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <inttypes.h>

#define PORT 8896

//defining the max charc size
#define MAX_CHARACTER_SIZE 32
#define MAX_TIME_ANCHOR 32
#define C 300000000
#define PACKET_LENGTH 1984

//the encrypt function
int encrypt(
    void* buffer,
    int buffer_len, /* Because the plaintext could include null bytes*/
    char* IV, //initilization vector
    char* key,
    int key_len);

//cipher text displyer
void printEncryptedFile(char* ciphertext,
                        int len,
                        FILE* encyptFile);

//Audio->AES
void audioToAESConversion(char const * pass);
```

```
//create the Tx values
void txValueFromKey(char const * key, char const * d1,
                    char const * d2, char const * d3);

//sends the TCP value
void TCPServiceRoutine();

//Creates Password from AES
void passwordCreation(char const * key, char * pass);


int main(int argc, char const *argv[])
{
          //check to see if all the argv is entred
    if (argc != 5)
    {
      printf("Usage: ./server <password> <d1> <d2> <d3>\n");
      return 0;
    }

    audioToAESConversion(argv[1]);

    txValueFromKey(argv[1], argv[2], argv[3], argv[4]);

    TCPServiceRoutine();

    return 0;
}

//the function that creates the AUDIO->AES
void audioToAESConversion(char const * key)
{
   //Step 1: Take Audio file -> Encypt the signal
    printf("\n==Location-Dependent Algorithm==\n");
    printf("\nEncryption Process Started\n");

    //open the respective audio and audio_text input files
    //ffmpeg -i pc.mp3 input.wav
    //ffmpeg -ss 2 -to 10 -i input.wav output.wav
    FILE * inputAudiofile;
    inputAudiofile  = popen("ffmpeg -i input.wav -hide_banner -f s16le -ac 1
-", "r");

    //setup AES Encrypt parameter
    //open the audio_text input and the encrypted file
    FILE * encyptFileOutput;
    encyptFileOutput = fopen("EncrFileOutput.txt","w");

    //create a MCRYPT to get certain info
    MCRYPT td = mcrypt_module_open("rijndael-256", NULL, "cbc", NULL);

    //A random block should be placed as the first block (IV)
    //so the same block or messages always encrypt to something different.
    char * IVEncr = malloc(mcrypt_enc_get_iv_size(td)); //return 8
    FILE * fp;
    fp = fopen("/dev/urandom", "r");
```

```
    fread(IVEncr, 1, mcrypt_enc_get_iv_size(td), fp);
    fclose(fp);
    mcrypt_generic_end(td);
    //place the IV in the encrypted file
    printEncryptedFile(IVEncr , mcrypt_enc_get_iv_size(td) ,
encyptFileOutput);

    //Generate the password
    unsigned char *obuf = SHA256(key, strlen(key), 0);
    char * pass = calloc(1, SHA256_DIGEST_LENGTH);

    printf("AES-258: %s : ", key);
    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
      pass[i] = obuf[i];
      printf("%02x", obuf[i]);
    }
    printf("\n");

    //check to see if the key is MAX_CHARACTER_SIZE charcter long
    char * keyEncr = calloc(1, MAX_CHARACTER_SIZE); //MAX_CHARACTER_SIZE * 8
= 128
    strncpy(keyEncr, pass, MAX_CHARACTER_SIZE);
    int keyEncrsize = MAX_CHARACTER_SIZE; /* 256 bits */

    //initialize the buffer
    int bufferEncr_len = MAX_CHARACTER_SIZE;
    char * bufferEncr = calloc(1, bufferEncr_len);

    //Encryption algorithm info display
    printf("The IV is: %s\n", IVEncr);
    printf("The Key is: %s\n\n", keyEncr);

    while(fread(bufferEncr, 4, 1, inputAudiofile)) // read one 4-byte sample
    {
      encrypt(bufferEncr, bufferEncr_len, IVEncr, keyEncr, keyEncrsize);
      printEncryptedFile(bufferEncr , bufferEncr_len , encyptFileOutput);
    }
    printf("Encryption Process Completed\n\n");

    //closing the pipe and file and freeing memory space
    pclose(inputAudiofile);
    fclose(encyptFileOutput);
    free(IVEncr);
    free(bufferEncr);
    free(keyEncr);
}

//the function that creates the Tx values
void txValueFromKey(char const * key,char const * d1, char const * d2, char
const * d3)
{
    printf("\nTx Value Creation starts \n");

    //input buffer
    char * ibuf = calloc(1, MAX_CHARACTER_SIZE);
    strncpy(ibuf, key, MAX_CHARACTER_SIZE);
```

```
    printf("\nPassword: %s\n", ibuf);
     //output buffer
    unsigned char *obuf = SHA256(ibuf, strlen(ibuf), 0);

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        printf("%02x", obuf[i]);
    }
    printf("\n\n");

    //create the Tx array
    int oTx [MAX_CHARACTER_SIZE] = {'0'};
    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        oTx[i] = obuf[i];
    }

    int dA = atoi(d1);
    int dB = atoi(d2);
    int dC = atoi(d3);

    srand(time(0));

    //Timing Variables
    uint64_t Tnet = 97500 * 65536;
    uint64_t TstartOffset = 0.005 * 975000 * 65536; //time for all the net
pakt to go 5ms
                            //319488000 (NT ticks)
    uint64_t TbtwnOffset = 0.0025 * 975000 * 65536; //time between each of my
seg
                            //159744000 (NT ticks)
    uint64_t Tslot = 0.00000000667 * 975000 * 65536; //time width of each
value
                            //of key 6.67nsec 300 NT ticks
    uint64_t TdistA = (dA/C) * 975000 * 65536 ;
    uint64_t TdistB = (dB/C) * 975000 * 65536;
    uint64_t TdistC = (dC/C) * 975000 * 65536;

    uint64_t Tdistlast = 0;
    uint64_t Ttxlast = 0;

    uint64_t Tx = 0;
    uint64_t Txfirst = 0;

    //first transmission being sent

    int Slot = 0;
    printf("The Sequence of Transmission\n");

    FILE * transmissionFile;
    transmissionFile = fopen("transmission.dat","w");

    //first transmission
    Txfirst = Tnet + TstartOffset;
    fprintf(transmissionFile, "%lu\n", Txfirst);
```

```
    FILE * dataFile;
    dataFile = fopen("EncrFileOutput.txt","r");
    int i = 0, breakflag = 0, data = 0, numPak = 0;

    FILE * debugFile;
    debugFile = fopen("Debug.txt","w");

    int bufferNumOld = 0, bufferNum = 0;

    while(1)
    {
      bufferNumOld = bufferNum;

      for (int j = 0; j < PACKET_LENGTH; j++)
      {
        if (fscanf(dataFile, "%d", &data) != EOF)
        {
          bufferNum++;
        }
        else
        {
          breakflag = 1;
          break;
        }
      }

      if (i < 32)
      {
        Slot = oTx[i];

        if (rand() % 2 == 0)
        {
          Tx = Ttxlast + Tdistlast + TbtwnOffset - TdistA + ((Slot + 0.5) *
Tslot);
          Tdistlast = TdistA;
          printf("AES: %02x Slot: %d Anchor A: %lu \n", oTx[i], oTx[i], Tx);
          fprintf(transmissionFile, "%d ", 0);

          fprintf(debugFile, "Ach#: %d ", 0);
          fprintf(debugFile, "Slot: %-5d ", Slot);
          fprintf(debugFile, "Tslot+d: %-10d ", (int)(TdistA + ((Slot + 0.5)
* Tslot)));
        }
        else if (rand() % 3 == 0)
        {
          Tx = Ttxlast + Tdistlast + TbtwnOffset - TdistB + ((Slot + 0.5) *
Tslot);
          Tdistlast = TdistB;
          printf("AES: %02x Slot: %d Anchor B: %lu \n", oTx[i], oTx[i], Tx);
          fprintf(transmissionFile, "%d ", 1);

          fprintf(debugFile, "Ach#: %d ", 1);
          fprintf(debugFile, "Slot: %-5d ", Slot);
          fprintf(debugFile, "Tslot+d: %-10d ", (int)(TdistB + ((Slot + 0.5)
* Tslot)));
        }
        else
```

```
        {
          Tx = Ttxlast + Tdistlast + TbtwnOffset - TdistC + ((Slot + 0.5) *
Tslot);
          Tdistlast = TdistC;
          printf("AES: %02x Slot: %d Anchor C: %lu \n", oTx[i], oTx[i], Tx);
          fprintf(transmissionFile, "%d ", 2);

          fprintf(debugFile, "Ach#: %d ", 2);
          fprintf(debugFile, "Slot: %-5d ", Slot);
          fprintf(debugFile, "Tslot+d: %-10d ", (int)(TdistC + ((Slot + 0.5)
* Tslot)));
        }

        fprintf(transmissionFile, "%lu\n",Tx);

        fprintf(debugFile, "1stB: %-10d LastB: %-10d\n",bufferNumOld,
bufferNum);

        Ttxlast = Tx;
        i++;
        numPak++;

        if(i == 32)
        {
          i = 0;
        }
      }

      if(breakflag)
      {
        break;
      }

    }


    printf("Num Pak:%d\n",numPak);
    fclose(transmissionFile);
    fclose(dataFile);
    fclose(debugFile);
    printf("\nTx value have been created\n");
}

//the function sends the TCP packets
void TCPServiceRoutine()
{
  printf("\nReady to Transmit Data.\n");
  int server_fd, new_socket, valread;
  struct sockaddr_in address;
  int opt = 1;
  int addrlen = sizeof(address);
  char buffer[1024] = {0};

  // Creating socket file descriptor
  if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
  {
      perror("socket failed");
```

```
    exit(EXIT_FAILURE);
}

// Forcefully attaching socket to the port 8080
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                                              &opt, sizeof(opt)))
{
    perror("setsockopt");
    exit(EXIT_FAILURE);
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( PORT );

// Forcefully attaching socket to the port 8080
if (bind(server_fd, (struct sockaddr *)&address,
                             sizeof(address))<0)
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}
if (listen(server_fd, 3) < 0)
{
    perror("listen");
    exit(EXIT_FAILURE);
}
if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                (socklen_t*)&addrlen))<0)
{
    perror("accept");
    exit(EXIT_FAILURE);
}

valread = read( new_socket , buffer, 1024);
printf("%s\n",buffer );

int counter = 0;

char packet [3000] = {0};
memset(packet, 0, sizeof(char) * 3000);
FILE * txFile;
txFile = fopen("transmission.dat","r");
FILE * dataFile;
dataFile = fopen("EncrFileOutput.txt","r");

uint64_t anchorNumber = 0;
uint64_t Tx = 0, Txstart = 0, firstByteNumber = 0;
int data = 0;

char anchorNumberbuff[21]= {0};
char firstByteNumberbuff[21]= {0};
char txBuff[21]= {0};
char dataBuff[PACKET_LENGTH]={0};

int breakflag = 0;

fscanf(txFile, "%lu" , &Txstart);
```

```
  while (read( new_socket , buffer, 1024) > 1)
  {
      memset(packet, 0, sizeof(char) * 3000);

      printf("%s\n",buffer);

      for (int i = 0; i < PACKET_LENGTH; i++)
      {
        if(fscanf(dataFile, "%d", &data) != EOF)
        {
          dataBuff[i] = data;
        }
        else
        {
          breakflag = 1;
          break;
        }

      }

      fscanf(txFile, "%lu" , &anchorNumber);
      sprintf(anchorNumberbuff, "%" PRIu64, anchorNumber);

      sprintf(firstByteNumberbuff, "%" PRIu64, firstByteNumber);

      fscanf(txFile, "%lu" , &Tx);
      sprintf(txBuff, "%" PRIu64, Tx);

      memcpy(&packet[0], anchorNumberbuff, sizeof(anchorNumberbuff));
      memcpy(&packet[22], firstByteNumberbuff, sizeof(firstByteNumberbuff));
      memcpy(&packet[50], txBuff, sizeof(txBuff));
      memcpy(&packet[100], dataBuff, sizeof(dataBuff));

      printf("\n\nPacket Content: AcNum:%s firstByte:%s Tx:%s Counter:%d\n",
&packet[0],
       &packet[22], &packet[50], counter);
      /*printf(" Data: ");
      for(int i = 0; i < PACKET_LENGTH; i++){printf("%d ", packet[100+i]);}
      printf("\n");*/

      send(new_socket , packet , sizeof(packet) , 0 );
      printf("Message sent\n");

      counter++;
      firstByteNumber += PACKET_LENGTH;
      if (firstByteNumber > 32 * PACKET_LENGTH){firstByteNumber = 0;}

      memset(buffer, 0, sizeof(char) * 1024);
      memset(anchorNumberbuff, 0, sizeof(char) * 21);
      memset(firstByteNumberbuff, 0, sizeof(char) * 21);
      memset(txBuff, 0, sizeof(char) * 21);
      memset(dataBuff, 0, sizeof(char) * PACKET_LENGTH);

      if (breakflag == 1)
      {
        break;
```

```
      }
  }

  printf("\n\nNumber of Packet sent: %d\n", counter );

  fclose(txFile);
  fclose(dataFile);
}

int encrypt(void* buffer, int buffer_len, char* IV, char* key, int key_len)
{
  MCRYPT td = mcrypt_module_open("rijndael-256", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  if( buffer_len % blocksize != 0 )
    {
      return 1;
    }

  mcrypt_generic_init(td, key, key_len, IV);
  mcrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
  mcrypt_module_close(td);

  return 0;
}

//displays as well as writes the encrypt file
void printEncryptedFile(char* ciphertext, int len, FILE* encyptFile)
{
  int v;
  for (v=0; v<len; v++)
  {
    fprintf(encyptFile, "%d", ciphertext[v]);
    fprintf(encyptFile, "%s", " ");
  }

  fprintf(encyptFile, "%s", "\n");
}
```

ii. TCP Client

```
// File: Makefile
// Class: EE 495-Senior Reseach            # Instructor: Mr. Mike Ciholas and
Dr. Donald Roberts
// Assignment: Location-Dept Cryptosystem  # Date started: 3/5/2018
//Programmer: Kunal Mukherjee              # Date completed:

//adding the header files for AES
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <mcrypt.h> //http://linux.die.net/man/3/mcrypt

//header file for Tx Maker
#include <openssl/sha.h>
#include <time.h>

//header file for TCP
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <inttypes.h>

#define PORT 8896

//defining the max charc size
#define MAX_CHARACTER_SIZE 32
#define MAX_TIME_ANCHOR 32
#define C 300000000
#define PACKET_LENGTH 1984


//the decrypt function
int decrypt(
    void* buffer,
    int buffer_len,
    char* IV,
    char* key,
    int key_len
);

int tcpServiceRoutine();

void aesToAudioConversion(char const * pass);

void keyFromTxValue(char * password);

//cipher text displyer
void printEncryptedFile(char* ciphertext,
                        int len,
                        FILE* encyptFile);
```

```
void playSong ();

int64_t S64(const char *s);

int main(int argc, char const *argv[])
{
    //check to see if all the argv is entred
    if (argc != 1)
    {
      printf("Usage: ./client\n");
      return 0;
    }
    //local variable password storage
    char * password = calloc(1, SHA256_DIGEST_LENGTH);

    printf("\n==Location-Dependent Algorithm==\n");

    tcpServiceRoutine();

    keyFromTxValue(password);

    aesToAudioConversion(password);

    printf("\n*** Location-Dependent Algorithm Process Completed ***\n\n");

    playSong();

    free(password);
    return 0;
}

int tcpServiceRoutine()
{
    int sock = 0;
    struct sockaddr_in serv_addr;
    char *hello = "Hello from client";
    char buffer[3000] = {0};

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    memset(&serv_addr, '0', sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
```

```
    {
        printf("\nConnection Failed \n");
        return -1;
    }

    send(sock , hello , strlen(hello) , 0 );
    printf("Ready to receive\n");

    char * received = "Packet acknowledgement received";

    send(sock , received , strlen(received) , 0 );
    int counter = 0;

    uint64_t anchorNumber = 0;
    uint64_t firstByteNumber = 0;
    uint64_t tx = 0;

    char anchorNumberbuff[21]= {0};
    char firstByteNumberbuff[21]= {0};
    char txBuff[21]= {0};
    char dataBuff[PACKET_LENGTH]= {0};

    int index = 0;

    FILE * receptionFile;
    receptionFile = fopen("receptionTx.dat","w");

    FILE * encrFile;
    encrFile = fopen("EncrFileOutput.txt","w");

    while (read( sock , buffer, 3000) > 1)
    {
        printf("\n\nPacket Content: AcNum:%s First Byte:%s Tx:%s
Packet:%d\n",
          &buffer[0], &buffer[22], &buffer[50], counter);
        /*printf(" Data: ");
        for(int i = 0; i < PACKET_LENGTH; i++){printf("%d ", buffer[100+i]);}
        printf("\n");*/

        index = 0;
        for(int i = 0;  i < 0 + 22; i++)
            {anchorNumberbuff[index] = buffer[i]; index++;}
        index = 0;
        for(int i = 22;  i < 22 + 22; i++)
            {firstByteNumberbuff[index] = buffer[i]; index++;}
        index = 0;
        for(int i = 50; i < 50 + 22; i++)
            {txBuff[index] = buffer[i]; index++;}
        index = 0;

        for (int i = 100; i < 100 + PACKET_LENGTH; i++)
            {dataBuff[index] = buffer[i]; index++;}
        index = 0;

        anchorNumber =  S64(anchorNumberbuff);
        firstByteNumber = S64(firstByteNumberbuff);
        tx = S64(txBuff);
```

```
        printf("Anchor Number: %lu\n", anchorNumber);
        printf("First Byte Number: %lu\n", firstByteNumber);
        printf("Tx: %lu\n", tx);
        printf("First Byte Num: %lu\n", tx);


        fprintf(receptionFile, "%lu ", anchorNumber);
        fprintf(receptionFile, "%lu ",tx);
        fprintf(receptionFile, "%lu\n", firstByteNumber);


        for(int i = 0; i < PACKET_LENGTH; i++)
          {fprintf(encrFile, "%d ", buffer[100+i]);}
        fprintf(encrFile, "%s", "\n");

        send(sock , received , strlen(received) , 0 );

        counter++;

        memset(buffer, 0, sizeof(char) * 3000);
        memset(anchorNumberbuff, 0, sizeof(char) * 21);
        memset(firstByteNumberbuff, 0, sizeof(char) * 21);
        memset(txBuff, 0, sizeof(char) * 21);
        memset(dataBuff,0, sizeof(char) * PACKET_LENGTH);
    }

    printf("\n\nNumber of Packet sent: %d\n", counter );

    fclose(receptionFile);
    fclose(encrFile);
    return 0;
}

void keyFromTxValue(char * password)
{
    printf("\nThe Key extraction Process Started\n");

    FILE * transmissionFile;
    transmissionFile = fopen("receptionTx.dat","r");

    int anchorNumber = 0;
    int receptionTime[MAX_TIME_ANCHOR] = {'0'};
    uint64_t Trx = 0, firstByteNumber = 0;
    int receptionTimeindex = 0;

    uint64_t TbtwnOffset = 0.0025 * 975000 * 65536;
    uint64_t Trxlast = 0;
    uint64_t Tslot = 0.00000000667 * 975000 * 65536; //time width of each
value of
                                                     //key 6.67nsec 426 NT
ticks
    uint64_t Tnoise = 0;
    srand(time(0));
    int margin = 120 ;   //60

    FILE * debugFile;
```

```
    debugFile = fopen("Debug.txt","w");
    int bufferNum = 0;

    while(fscanf(transmissionFile, "%d" , &anchorNumber) != EOF)
    {
        fscanf(transmissionFile, "%lu" , &Trx);
        fscanf(transmissionFile, "%lu" , &firstByteNumber);

        Tnoise = (rand() % (2 * ((Tslot/2) - margin + 1))) - ((Tslot/2) -
margin + 1);

        if(bufferNum < 32)
        {
          receptionTime[receptionTimeindex] =
                       (Trx + Tnoise - Trxlast - TbtwnOffset) / Tslot;
        }

        fprintf(debugFile, "Ach: %-5d ", anchorNumber);
        fprintf(debugFile, "Tslot: %-5lu ", ((Trx + Tnoise - Trxlast -
TbtwnOffset) / Tslot));
        fprintf(debugFile, "Tnoise: %-5d ", (int)Tnoise);
        fprintf(debugFile, "Trx+n: %-15lu ", (Trx + Tnoise));
        fprintf(debugFile, "Trxlast: %-15lu ", Trxlast);
        fprintf(debugFile, "TbtwnOffset: %-15lu ", TbtwnOffset);
        fprintf(debugFile, "IstB %-10lu\n", firstByteNumber);

        Trxlast = Trx + Tnoise;
        receptionTimeindex++;
        bufferNum++;
    }

    printf("The Key Is: \n");
    unsigned char * oBuf = calloc(1, MAX_CHARACTER_SIZE);

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        oBuf[i] = receptionTime[i];
    }

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        printf("%02x", oBuf[i]);
    }
    printf("\n");

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
      password[i] = oBuf[i];;
    }

    fclose(transmissionFile);
    fclose(debugFile);
    printf("\nThe Key extraction Process Ended\n");
}

void aesToAudioConversion(char const * password)
{
```

```
  //decryption algorithm
  printf("Decryption Process Started\n");
  printf("Converting the decrypted audio sample to audio file\n");

  //open the encrypted file to get the IV and the AES blocks
  //open the output_digitalized text that would be created
  FILE * encyptFileInput;
  encyptFileInput = fopen("EncrFileOutput.txt","r");

  //initialize decrypt buffer
  int bufferDecr_len = MAX_CHARACTER_SIZE;
  char * bufferDecr = calloc(1, bufferDecr_len);

  //initialize the key
  char * keyDecr = calloc(1, MAX_CHARACTER_SIZE); //MAX_CHARACTER_SIZE * 8 =
128
  strncpy(keyDecr, password, MAX_CHARACTER_SIZE);
  int keyDecrsize = MAX_CHARACTER_SIZE;

  //getting IV
  char * IVDecr = calloc(1, MAX_CHARACTER_SIZE);
  int IVBuf = 0;

  for (int i = 0; i < MAX_CHARACTER_SIZE; i++)
  {
    fscanf(encyptFileInput, "%d" , &IVBuf);
    IVDecr[i] = IVBuf;
  }

  //display the decrypting info
  printf("The IV is: %s\n", IVDecr);
  printf("The Key is: %s\n", keyDecr);

  //initilizing he tempAESbuffer and bufferIndex
  int  AESbuf = 0;
  int bufIndex = 0;

  //opening a output_digitalized file to create the audio
  FILE * outputAudiofile;
  outputAudiofile = popen("ffmpeg -y -f s16le -ar 44100 -ac 1 -i - output.wav
-hide_banner", "w");

  //getting the AES blocks from the encrypt_file and decrypting it
  while(fscanf(encyptFileInput, "%d" , &AESbuf) != EOF)
  {
    if(bufIndex < MAX_CHARACTER_SIZE)
    {
      bufferDecr[bufIndex] = AESbuf;
      bufIndex++;
    }
    else
    {
      decrypt(bufferDecr, bufferDecr_len, IVDecr, keyDecr, keyDecrsize);
      fwrite(bufferDecr, 4, 1, outputAudiofile);

      bufIndex = 0;
      memset(bufferDecr, 0 , MAX_CHARACTER_SIZE);
```

```
      bufferDecr[bufIndex] = AESbuf;
      bufIndex++;
    }
  }

  if( strlen(bufferDecr) != 0)
  {
    decrypt(bufferDecr, bufferDecr_len, IVDecr, keyDecr, keyDecrsize);
    fwrite(bufferDecr, 4, 1, outputAudiofile);
  }

  printf("Decryption Process Completed and Audio File Made\n");

  //closing the output file and free memory
  fclose(encyptFileInput);

  free(keyDecr);
  free(bufferDecr);
  free(IVDecr);

  pclose(outputAudiofile);
}

int decrypt(void* buffer, int buffer_len, char* IV, char* key, int key_len)
{
  MCRYPT td = mcrypt_module_open("rijndael-256", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  if( buffer_len % blocksize != 0 )
    {
      return 1;
    }

  mcrypt_generic_init(td, key, key_len, IV);
  mdecrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
  mcrypt_module_close(td);

  return 0;
}

int64_t S64(const char *s)
{
  int64_t i;
  char c ;
  int scanned = sscanf(s, "%"  PRIu64 "%c", &i, &c);
  if (scanned == 1) return i;
  if (scanned > 1) {
    // TBD about extra data found
    return i;
    }
  // TBD has failed to scan;
  return 0;
}

void playSong(void)
{
  //play the resultent audio
```

```
  FILE * playResultAudio;
  playResultAudio = popen("ffplay -hide_banner -autoexit output.wav", "r");
  pclose (playResultAudio);
}
```

Appendix D. Location Dependent Cryptosystem

i. *Encryption Script*

```
// File: Makefile
// Class: EE 495-Senior Reseach            # Instructor: Mr. Mike Ciholas and
Dr. Donald Roberts
// Assignment: Location-Dept Cryptosystem  # Date started: 3/29/2018
//Programmer: Kunal Mukherjee              # Date completed:

//adding the header files for encryption
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <mcrypt.h> //http://linux.die.net/man/3/mcrypt
//sudo apt-get install libmcrypt-dev
//sudo apt-get install ffmpeg

//header for Tx values
//sudo apt-get install libssl-dev
#include <openssl/sha.h>
#include <time.h>

//defining the max charc size
#define MAX_CHARACTER_SIZE 32
#define MAX_TIME_ANCHOR 32
#define C 300000000
#define PACKET_LENGTH 800

//the encrypt function
int encrypt(
    void* buffer,
    int buffer_len, /* Because the plaintext could include null bytes*/
    char* IV, //initilization vector
    char* key,
    int key_len);

//cipher text displyer
void printEncryptedFile(char* ciphertext,
                        int len,
                        FILE* encyptFile);

//Audio->AES
void audioToAESConversion(char const * pass);

//create the Tx values
void theAESKey(char const * key);

int main(int argc, char const *argv[])
{
          //check to see if all the argv is entred
    if (argc != 2)
    {
      printf("Usage: ./server <password>\n");
```

```
      return 0;
    }

    audioToAESConversion(argv[1]);

    theAESKey(argv[1]);

    return 0;
}

//the function that creates the AUDIO->AES
void audioToAESConversion(char const * key)
{
   //Step 1: Take Audio file -> Encypt the signal
    printf("\n==Location-Dependent Algorithm==\n");
    printf("\nEncryption Process Started\n");

    //open the respective audio and audio_text input files
    //ffmpeg -i pc.mp3 input.wav
    //ffmpeg -ss 2 -to 10 -i input.wav output.wav
    FILE * inputAudiofile;
    inputAudiofile  = popen("ffmpeg -i input.wav -hide_banner -f s16le -ac 1
-", "r");

    //setup AES Encrypt parameter
    //open the audio_text input and the encrypted file
    FILE * encyptFileOutput;
    encyptFileOutput = fopen("EncrFileOutput.txt","w");

    //create a MCRYPT to get certain info
    MCRYPT td = mcrypt_module_open("rijndael-256", NULL, "cbc", NULL);

    //A random block should be placed as the first block (IV)
    //so the same block or messages always encrypt to something different.
    char * IVEncr = malloc(mcrypt_enc_get_iv_size(td)); //return 8
    FILE * fp;
    fp = fopen("/dev/urandom", "r");
    fread(IVEncr, 1, mcrypt_enc_get_iv_size(td), fp);
    fclose(fp);
    mcrypt_generic_end(td);
    //place the IV in the encrypted file
    printEncryptedFile(IVEncr , mcrypt_enc_get_iv_size(td) ,
encyptFileOutput);

    //Generate the password
    unsigned char *obuf = SHA256(key, strlen(key), 0);
    char * pass = calloc(1, SHA256_DIGEST_LENGTH);

    printf("AES-258: %s : ", key);
    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
      pass[i] = obuf[i];
      printf("%02x", obuf[i]);
    }
    printf("\n");

    //check to see if the key is MAX_CHARACTER_SIZE charcter long
```

```
    char * keyEncr = calloc(1, MAX_CHARACTER_SIZE); //MAX_CHARACTER_SIZE * 8
= 128
    strncpy(keyEncr, pass, MAX_CHARACTER_SIZE);
    int keyEncrsize = MAX_CHARACTER_SIZE; /* 256 bits */

    //initialize the buffer
    int bufferEncr_len = MAX_CHARACTER_SIZE;
    char * bufferEncr = calloc(1, bufferEncr_len);

    //Encryption algorithm info display
    printf("The IV is: %s\n", IVEncr);
    printf("The Key is: %s\n\n", keyEncr);

    while(fread(bufferEncr, 4, 1, inputAudiofile)) // read one 4-byte sample
    {
      encrypt(bufferEncr, bufferEncr_len, IVEncr, keyEncr, keyEncrsize);
      printEncryptedFile(bufferEncr , bufferEncr_len , encyptFileOutput);
    }
    printf("Encryption Process Completed\n\n");

    //closing the pipe and file and freeing memory space
    pclose(inputAudiofile);
    fclose(encyptFileOutput);
    free(IVEncr);
    free(bufferEncr);
    free(keyEncr);
}

//the function that prints the AES value
void theAESKey(char const * key)
{
        printf("\nThe AES Code\n");

    //input buffer
    char * ibuf = calloc(1, MAX_CHARACTER_SIZE);
    strncpy(ibuf, key, MAX_CHARACTER_SIZE);

    printf("\nPassword: %s\n", ibuf);
     //output buffer
    unsigned char *obuf = SHA256(ibuf, strlen(ibuf), 0);

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        printf("%02x", obuf[i]);
    }
    printf("\n\n");

    FILE * transmissionFile;
    transmissionFile = fopen("key.dat","w");

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        fprintf(transmissionFile, "%02x ", obuf[i]);
    }

    fclose(transmissionFile);
}
```

```
int encrypt(void* buffer, int buffer_len, char* IV, char* key, int key_len)
{
  MCRYPT td = mcrypt_module_open("rijndael-256", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  if( buffer_len % blocksize != 0 )
    {
      return 1;
    }

  mcrypt_generic_init(td, key, key_len, IV);
  mcrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
  mcrypt_module_close(td);

  return 0;
}

//displays as well as writes the encrypt file
void printEncryptedFile(char* ciphertext, int len, FILE* encyptFile)
{
  int v;
  for (v=0; v<len; v++)
  {
    fprintf(encyptFile, "%d", ciphertext[v]);
    fprintf(encyptFile, "%s", " ");
  }

  fprintf(encyptFile, "%s", "\n");
}
```

ii. Decryption Script

```
// File: Makefile
// Class: EE 495-Senior Reseach            # Instructor: Mr. Mike Ciholas and
Dr. Donald Roberts
// Assignment: Location-Dept Cryptosystem  # Date started: 3/5/2018
//Programmer: Kunal Mukherjee              # Date completed:

//adding the header files for AES
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <mcrypt.h> //http://linux.die.net/man/3/mcrypt

//header file for Tx Maker
#include <openssl/sha.h>
#include <time.h>

//defining the max charc size
#define MAX_CHARACTER_SIZE 32
#define MAX_TIME_ANCHOR 32
#define C 300000000
#define PACKET_LENGTH 1984
#define KEY_FILE "key.dat"//"receptionTx.dat"


//the decrypt function
int decrypt(
    void* buffer,
    int buffer_len,
    char* IV,
    char* key,
    int key_len
);

void aesToAudioConversion(char const * pass);

void keyFromFile(char * password);

//cipher text displyer
void printEncryptedFile(char* ciphertext,
                        int len,
                        FILE* encyptFile);

void playSong ();

int64_t S64(const char *s);

int main(int argc, char const *argv[])
{
    //check to see if all the argv is entred
    if (argc != 1)
    {
```

```
      printf("Usage: ./client\n");
      return 0;
    }
    //local variable password storage
    char * password = calloc(1, SHA256_DIGEST_LENGTH);

    printf("\n==Location-Dependent Algorithm==\n");

    keyFromFile(password);

    aesToAudioConversion(password);

    printf("\n*** Location-Dependent Algorithm Process Completed ***\n\n");

    playSong();

    free(password);
    return 0;
}

void keyFromFile(char * password)
{
    FILE * transmissionFile;
    transmissionFile = fopen(KEY_FILE,"r");

    int receptionTime[MAX_TIME_ANCHOR] = {'0'};
    int receptionTimeindex = 0;

    int keyValue = 0;

    while(fscanf(transmissionFile, "%x" , &keyValue) != EOF)
    {
        receptionTime[receptionTimeindex] = keyValue;
        receptionTimeindex++;
    }

    printf("The Key Is: \n");
    unsigned char * oBuf = calloc(1, MAX_CHARACTER_SIZE);

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        oBuf[i] = receptionTime[i];
    }

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        printf("%02x ", oBuf[i]);
    }
    printf("\n");

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
      password[i] = oBuf[i];;
    }

    fclose(transmissionFile);
}
```

```
void aesToAudioConversion(char const * password)
{
  //decryption algorithm
  printf("Decryption Process Started\n");
  printf("Converting the decrypted audio sample to audio file\n");

  //open the encrypted file to get the IV and the AES blocks
  //open the output_digitalized text that would be created
  FILE * encyptFileInput;
  encyptFileInput = fopen("EncrFileOutput.txt","r");

  //initialize decrypt buffer
  int bufferDecr_len = MAX_CHARACTER_SIZE;
  char * bufferDecr = calloc(1, bufferDecr_len);

  //initialize the key
  char * keyDecr = calloc(1, MAX_CHARACTER_SIZE); //MAX_CHARACTER_SIZE * 8 =
128
  strncpy(keyDecr, password, MAX_CHARACTER_SIZE);
  int keyDecrsize = MAX_CHARACTER_SIZE;

  //getting IV
  char * IVDecr = calloc(1, MAX_CHARACTER_SIZE);
  int IVBuf = 0;

  for (int i = 0; i < MAX_CHARACTER_SIZE; i++)
  {
    fscanf(encyptFileInput, "%d" , &IVBuf);
    IVDecr[i] = IVBuf;
  }

  //display the decrypting info
  printf("The IV is: %s\n", IVDecr);
  printf("The Key is: %s\n", keyDecr);

  //initilizing he tempAESbuffer and bufferIndex
  int  AESbuf = 0;
  int bufIndex = 0;

  //opening a output_digitalized file to create the audio
  FILE * outputAudiofile;
  outputAudiofile = popen("ffmpeg -y -f s16le -ar 44100 -ac 1 -i - output.wav
-hide_banner", "w");

  //getting the AES blocks from the encrypt_file and decrypting it
  while(fscanf(encyptFileInput, "%d" , &AESbuf) != EOF)
  {
    if(bufIndex < MAX_CHARACTER_SIZE)
    {
      bufferDecr[bufIndex] = AESbuf;
      bufIndex++;
    }
    else
    {
      decrypt(bufferDecr, bufferDecr_len, IVDecr, keyDecr, keyDecrsize);
      fwrite(bufferDecr, 4, 1, outputAudiofile);
```

```
      bufIndex = 0;
      memset(bufferDecr, 0 , MAX_CHARACTER_SIZE);
      bufferDecr[bufIndex] = AESbuf;
      bufIndex++;
    }
  }

  if( strlen(bufferDecr) != 0)
  {
    decrypt(bufferDecr, bufferDecr_len, IVDecr, keyDecr, keyDecrsize);
    fwrite(bufferDecr, 4, 1, outputAudiofile);
  }

  printf("Decryption Process Completed and Audio File Made\n");

  //closing the output file and free memory
  fclose(encyptFileInput);

  free(keyDecr);
  free(bufferDecr);
  free(IVDecr);

  pclose(outputAudiofile);
}

int decrypt(void* buffer, int buffer_len, char* IV, char* key, int key_len)
{
  MCRYPT td = mcrypt_module_open("rijndael-256", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  if( buffer_len % blocksize != 0 )
    {
      return 1;
    }

  mcrypt_generic_init(td, key, key_len, IV);
  mdecrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
  mcrypt_module_close(td);

  return 0;
}

int64_t S64(const char *s)
{
  int64_t i;
  char c ;
  int scanned = sscanf(s, "%"  PRIu64 "%c", &i, &c);
  if (scanned == 1) return i;
  if (scanned > 1) {
    // TBD about extra data found
    return i;
    }
  // TBD has failed to scan;
  return 0;
}
```

```
void playSong(void)
{
  //play the resultent audio
  FILE * playResultAudio;
  playResultAudio = popen("ffplay -hide_banner -autoexit output.wav", "r");
  pclose (playResultAudio);
}
```

iii. UWB-Sender

```
#!/usr/bin/env python3
import socket
from cdp import *
from cdp.ciholas_serial_number import *
import datetime
import time
import struct
import argparse
import os
import hashlib

parser = argparse.ArgumentParser(description = 'Pass in encryption file, send
cdp encryption packets, receive corrected encryption packets')
parser.add_argument('-filename', type = str, default =
'AES_App_for_NetApp/client/EncrFileOutput.txt', help='Path to encryption
file')

CONFIG_PORT = 7671
INTERNAL_PORT = 7694
interface = '10.99.130.193'
group = "239.255.76.94"

#10Mhz
REPEAT_RATE = 97500
#Ensures we're well clear of command window to start sending our own cdp
packets
SCHEDULE_TIME_OFFSET = 10000
#Skip ahead a frame or two to make sure there's enough time to schedule our
packet
CADENCE_OFFSET = 1
#CADENCE_OFFSET = 2

#Uncomminging first one is quicker transmission, but key comes out noisy.
Second one makes transmisison very slow, but improves accuracy
TICKS_PER_SECOND =  97500 * 65536
#TICKS_PER_SECOND =  975000 * 65536


#Speed of light in m/s
C = 300000000
#Distance of anchors from receivers in meter
ANCHOR_DISTANCE = 1

#How many nts to receive before attempting to send an encryption packet (how
often to send)
#Not being used
#TICK_COUNT_BEFORE_SENDING = 2
#TICK_COUNT_BEFORE_SENDING = 15
TICK_COUNT_BEFORE_SENDING = 150

#Addresses of anchors
ANCHOR_05AE_ADDRESS = ('10.99.130.136', 49153)
ANCHOR_0288_ADDRESS = ('10.99.129.182', 49153)
```

```
ANCHOR_0225_ADDRESS = ('10.99.130.159', 49153)
ANCHOR_05BB_ADDRESS = ('10.99.130.151', 49153)
ANCHOR_0409_ADDRESS = ('10.99.130.158', 49153)

#How many bits to shift nt64
SHIFT_AMOUNT = 16

#Encryption packets under this length will be padded with trailing spaces
(packets larger are left alone)
MIN_PACKET_SIZE = 20

#setup aggregation data read socket
data_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)
data_socket.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, 32)
data_socket.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_LOOP, 1)
data_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
data_socket.bind(('', INTERNAL_PORT))
#data_socket.bind(('', CONFIG_PORT))
data_socket.setsockopt(socket.SOL_IP, socket.IP_MULTICAST_IF,
socket.inet_aton(interface))
data_socket.setsockopt(socket.SOL_IP, socket.IP_ADD_MEMBERSHIP,
socket.inet_aton(group) + socket.inet_aton(interface))

accepted_serials = [0x01040288, 0x010405C9, 0x010405AE, 0x01040225,
0x010405BB, 0x01040409]

print('Sending packets on ', INTERNAL_PORT)

tickCount = 0
awaiting_reply = False
sentTime = None
lastSentPacket = None
receiver_addr = None
payload_strings = []
received_packets = []

#               0288,                   0409,                       05BB,
05AE
#ANCHORS = [ ('10.99.129.182', 49153), ('10.99.130.158', 49153),
('10.99.130.151', 49153), ('10.99.130.136', 49153) ]

ANCHORS = [ ANCHOR_05BB_ADDRESS, ANCHOR_0409_ADDRESS, ANCHOR_0225_ADDRESS,
ANCHOR_05AE_ADDRESS ]

ACNHOR_RECV = ANCHOR_05AE_ADDRESS

class EncryptionPacket(CDPDataItem):
    """CDP Data Item: Ciholas Data Protocol Acknowledge List Response Data
Item Definition"""

    type = 0x807F
    definition = [DIUInt64Attr('network_time'),
                  DIVariableLengthBytesAttr('data_payload'),]

class CorrectedEncryptionPacket(CDPDataItem):
```

```
    """CDP Data Item: Ciholas Data Protocol Acknowledge List Response Data
Item Definition"""

    type = 0x8080
    definition = [DIUInt64Attr('network_time'),
                  DIVariableLengthBytesAttr('data_payload'),]

class EncryptionPacketBuffer:
    anchorNumber = None
    nt64_send = None
    byte = None

    def __init__(self, anchor, time, byte):
        self.anchorNumber = anchor
        self.nt64_send = time
        self.byte = byte

#Read encryption file, return list of (nt, packet) tuples
def ReadEncryptionFile(fileName):
    script_dir = os.path.dirname(__file__) #<-- absolute dir the script is in
    abs_file_path = os.path.join(script_dir, fileName)

    file = open(abs_file_path)

    packet = []
    data = []
    lineIdx = 0


    BYTES_IN_LINE = 32
    LINES_IN_PACKET = 800 / BYTES_IN_LINE

    while True:
        lineIdx += 1
        line = file.readline()
        #Stop at EOF
        if(line == ''):
            #Add last packet before leaving
            data.append( packet )
            packet = []
            break
        #Read data
        else:
            line = line.rstrip()
            #List of ints
            lineData = list(map(int,line.split(' ')))

            #Convert int8 to uint8
            for idx,num in enumerate(lineData):
                if(num < 0):
                    lineData[idx] = num + 2**8

            packet += (lineData)
            #buffer = EncryptionPacketBuffer(int(lineData[0]),
int(lineData[1]), lineData[2])
        #Every 800 lines is 1 packet length
        if(lineIdx % LINES_IN_PACKET == 0):
```

```
            data.append( packet )
            packet = []


    file.close()

    return data

def SendEncryptionBuffer(timeToSend, bytesToSend, anchorToSend):
    global ANCHORS

    timeToSend = round(timeToSend)

    #Create data
    dataItem = EncryptionPacket()
    dataItem.network_time = timeToSend
    #dataItem.data_payload = byteToSend.ljust(MIN_PACKET_SIZE, ' ')
    dataItem.data_payload = bytes(bytesToSend)

    #Create packet, add header, data payload
    cdp_packet = CDP()
    #This serial number doesn't seem to matter?
    cdp_packet.serial_number = CiholasSerialNumber(0x01040225)
    cdp_packet.add_data_item(dataItem)
    packet_data = cdp_packet.encode()

    print(datetime.datetime.now().time(), ' - Sending encrypion packet with
nt: \t', hex(timeToSend))

    #send packet
    data_socket.sendto(packet_data, ANCHORS[anchorToSend])




args = parser.parse_args()

#Read file
packet_list = ReadEncryptionFile(args.filename)

dA = ANCHOR_DISTANCE
dB = ANCHOR_DISTANCE
dC = ANCHOR_DISTANCE


last_sent = None

#Distance of anchor to receiver
TdistA = (dA // C) * TICKS_PER_SECOND
TdistB = (dB // C) * TICKS_PER_SECOND
TdistC = (dC // C) * TICKS_PER_SECOND

anchorDistances = [TdistA, TdistB, TdistC]

#Frame offset
TbtwnOffset = .0025 * TICKS_PER_SECOND
```

```
#1/255 of space of 2m sphere
Tslot = 0.00000000667 * TICKS_PER_SECOND
#Tslot = 0.00000001668 * TICKS_PER_SECOND

#Distance of last anchor to receiver
Tdistlast = 0
#Time of last transmission
Ttxlast = 0
#Time to transmit
Tx = 0

TrxLast = 0

currentAnchor = 0

#First one is dummy packet
sha = [0x88, 0x88, 0x88, 0x88, 0x88,
0x5E,0x88,0x48,0x98,0xDA,0x28,0x04,0x71,0x51,0xD0,0xE5,0xC6,0x29,0x27,0x73,0x
60,0x3D,0x0D,0x6A,0xAB,0xBD,0xD6,0xEF,0x72,0x1D,0x15,0x42,0xD8,0x49,0x29,0xFF
,0x01]
shaIndex = 0

counter = TICK_COUNT_BEFORE_SENDING

packetIdx = 0

GARBAGE_DATA = [1,2,3,4,5,6,7,8,9,10]

try:
    while(packetIdx < len(packet_list)):
        while shaIndex < len(sha): # or awaiting_reply:
            try:
                #Read packets until we get a reply from the receiving anchor
                data, addr = data_socket.recvfrom(65536)
                new_rx_packet = CDP(data) #decode the data into a cdp packet
and decode data items into their appropriate types\

                #print( 'Serial: ', new_rx_packet.serial_number, '\t Addr: ',
addr )

                for data_item in new_rx_packet.data_items_by_type[0x802D]:
                    #if(not awaiting_reply):
                    if(counter > 0):
                        if(Ttxlast == 0):
                            startTime = data_item.nt64 // TICKS_PER_SECOND
                            startTime = startTime * TICKS_PER_SECOND

                            Ttxlast = data_item.nt64 + TICKS_PER_SECOND
                        else:
                            slot = sha[shaIndex]
                            shaIndex +=1

                            #Send packet, then wait for reply
                            #TICKS_PER_SECOND is a one frame offset
                            time_to_send = Ttxlast + Tdistlast + TbtwnOffset
- anchorDistances[currentAnchor] + ((slot + 0.5) * Tslot) + TICKS_PER_SECOND
```

```
                            print('\nCurrent nt: \t\t\t\t\t\t',
hex(data_item.nt64), 'ShaIndex: ', shaIndex-5)

                            packetsToSend = [shaIndex] +
packet_list[packetIdx] if shaIndex > 5 else GARBAGE_DATA

                            SendEncryptionBuffer( time_to_send,
packetsToSend, currentAnchor )
                            awaiting_reply = True

                            Ttxlast = time_to_send

                            Tdistlast = anchorDistances[currentAnchor]

                            currentAnchor += 1
                            if(currentAnchor > 2):
                                currentAnchor = 0

                            keyByte = slot

                            counter -= 1

                            packetIdx += 1
                    else:
                        counter = TICK_COUNT_BEFORE_SENDING




                # if(addr == ANCHORS[3]):
                #     for data_item in
new_rx_packet.data_items_by_type[0x8080]:
                #         print('got 0x8080 from serial:',
new_rx_packet.serial_number, ' nt: \t\t', hex(data_item.network_time))
                #         print ('payload: ', data_item.data_payload.strip()
)
                #         #print ('time received: ',
hex(data_item.network_time) )

                #         #Calculate keyByte
                #         keyByte = (data_item.network_time - TrxLast -
TbtwnOffset - TICKS_PER_SECOND) / Tslot

                #         #Update TrxLast for next recv
                #         TrxLast = data_item.network_time

                #         print('Key byte = ', hex(int(keyByte)), '\tSha
index: ', shaIndex)

                #         awaiting_reply = False

            except (KeyboardInterrupt, SystemExit):
                data_socket.close()

        shaIndex = 5
        print('Done with sha loop')
```

```
    print('Sent all packets')

except Exception as e:
    print (e)
```

iv. UWB-Receiver

```
#!/usr/bin/env python3

import socket
from cdp import *
#from cdp.ciholas_serial_number import *
import datetime
import time
import struct
import argparse
import os
import hashlib
import codecs
import binascii

parser = argparse.ArgumentParser(description = 'Pass in encryption file, send
cdp encryption packets, receive corrected encryption packets')
parser.add_argument('-filename', type = str, default = 'EncrFileOutput.txt',
help='Path to encryption file')

parserK = argparse.ArgumentParser(description = 'Pass in key file')
parserK.add_argument('-filename', type = str, default =
'Transmission/key.dat', help='Path to encryption file')


CONFIG_PORT = 7671
INTERNAL_PORT = 7694
#interface = '10.99.127.70'
interface = '10.99.127.70'
group = "239.255.76.94"

#Addresses of anchors
ANCHOR_0081_ADDRESS = ('10.99.129.105', 49153)
ANCHOR_02FE_ADDRESS = ('10.99.129.180', 49153)
ANCHOR_0249_ADDRESS = ('10.99.129.145', 49153)
ANCHOR_01E3_ADDRESS = ('10.99.130.195', 49153)

#How many bits to shift nt64
SHIFT_AMOUNT = 16

#Encryption packets under this length will be padded with trailing spaces
(packets larger are left alone)
MIN_PACKET_SIZE = 20

#Convert uint bytes to ints, then to string
def convertBuffer(buffer):
    intBuffer = []
    #uintFromByte = struct.unpack('>H', buffer)
    for item in buffer:
        intFromUInt = item if item <128 else item - 2**8
        intBuffer.append(intFromUInt)

    string = ' '.join(str(e) for e in intBuffer)
```

```
    return string

def convertKeyToHex(key):
    string = ''
    for item in key:
        string += str(hex(int(item)).lstrip("0x"))
        string += ' '

    return string

def writeToFile(stringToWrite, fileName):
    fileDes = open(fileName, 'w+')

    fileDes.write(stringToWrite)

    fileDes.close()

    return

#setup aggregation data read socket
data_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)
data_socket.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, 32)
data_socket.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_LOOP, 1)
data_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
data_socket.bind(('', INTERNAL_PORT))
#data_socket.bind(('', CONFIG_PORT))
data_socket.setsockopt(socket.SOL_IP, socket.IP_MULTICAST_IF,
socket.inet_aton(interface))
data_socket.setsockopt(socket.SOL_IP, socket.IP_ADD_MEMBERSHIP,
socket.inet_aton(group) + socket.inet_aton(interface))

accepted_serials = [0x01040081, 0x010402FE, 0x01040249, 0x010401E3]

print('Listening for packets on ', INTERNAL_PORT)

tickCount = 0
awaiting_reply = False
sentTime = None
lastSentPacket = None
receiver_addr = None
payload_strings = []
received_packets = []

ANCHORS = [ ANCHOR_0081_ADDRESS, ANCHOR_02FE_ADDRESS, ANCHOR_0249_ADDRESS,
ANCHOR_01E3_ADDRESS ]

ACNHOR_RECV = ANCHORS[3]

#variables for the cryptosystem
TICKS_PER_SECOND =  975000 * 65536
#TICKS_PER_SECOND =  97500 * 65536

#OFFSET = 975000 * 65536

#Speed of light in m/s
C = 300000000
```

```
#Distance of anchors from receivers in meter

#distances from the anchor
dA = 4.961
dB = 4.812
dC = 4.826

last_sent = None

#Distance of anchor to receiver
TdistA = (dA // C) * TICKS_PER_SECOND
TdistB = (dB // C) * TICKS_PER_SECOND
TdistC = (dC // C) * TICKS_PER_SECOND

anchorDistances = [TdistA, TdistB, TdistC]

#Frame offset
TbtwnOffset = .0025 * TICKS_PER_SECOND

#1/255 of space of 2m sphere
Tslot = 0.00000000667 * TICKS_PER_SECOND

#Distance of last anchor to receiver
Tdistlast = 0
#Time of last transmission
Ttxlast = 0
#Time to transmit
Tx = 0
#time of last reception
TrxLast = 0
#current anor selected
currentAnchor = 1

#First one is dummy packet
sha = []#[0xFF,
0x5E,0x88,0x48,0x98,0xDA,0x28,0x04,0x71,0x51,0xD0,0xE5,0xC6,0x29,0x27,0x73,0x
60,0x3D,0x0D,0x6A,0xAB,0xBD,0xD6,0xEF,0x72,0x1D,0x15,0x42,0xD8,0x49,0x29,0xFF
,0x01]
shaIndex = 0

#print("SHA LEN: ", len(sha))

#Received Key bytes
#key = [0 for i in range(0,33)]
key = []

#print("KEY LEN", len(key))

print(' ')

packetIdx = 0

#current time
curentTime = 0

class EncryptionPacket(CDPDataItem):
```

```
    """CDP Data Item: Ciholas Data Protocol Acknowledge List Response Data
Item Definition"""

    type = 0x807F
    definition = [DIUInt64Attr('network_time'),
                  DIVariableLengthBytesAttr('data_payload'),]

class CorrectedEncryptionPacket(CDPDataItem):
    """CDP Data Item: Ciholas Data Protocol Acknowledge List Response Data
Item Definition"""

    type = 0x8080
    definition = [DIUInt64Attr('network_time'),
                  DIVariableLengthBytesAttr('data_payload'),]

class EncryptionPacketBuffer:
    anchorNumber = None
    nt64_send = None
    byte = None

    def __init__(self, anchor, time, byte):
        self.anchorNumber = anchor
        self.nt64_send = time
        self.byte = byte

def ReadKeyFile(fileName):
    script_dir = os.path.dirname(__file__) #<-- absolute dir the script is in
    abs_file_path = os.path.join(script_dir, fileName)

    file = open(abs_file_path)

    BYTES_IN_LINE = 32

    data = []

    data.append(0xFF)

    while True:
        line = file.readline()
        #Stop at EOF
        if(line == ''):
            #Add last packet before leaving
            break
        #Read data
        else:
            line = line.rstrip()

            #print(line)
            #List of ints
            lineData = list(line.split(' '))
            #print(lineData)

            for item in lineData:
                data.append(int(item,16))

            #print (data)
```

```
    file.close()

    return data


#Read encryption file, return list of (nt, packet) tuples
def ReadEncryptionFile(fileName):
    script_dir = os.path.dirname(__file__) #<-- absolute dir the script is in
    abs_file_path = os.path.join(script_dir, fileName)

    file = open(abs_file_path)

    packet = []
    data = []
    lineIdx = 0


    BYTES_IN_LINE = 32
    LINES_IN_PACKET = 800 / BYTES_IN_LINE

    while True:
        lineIdx += 1
        line = file.readline()
        #Stop at EOF
        if(line == ''):
            #Add last packet before leaving
            data.append( packet )
            packet = []
            break
        #Read data
        else:
            line = line.rstrip()
            #List of ints
            lineData = list(map(int,line.split(' ')))

            #Convert int8 to uint8
            for idx,num in enumerate(lineData):
                if(num < 0):
                    lineData[idx] = num + 2**8

            packet += (lineData)
            buffer = EncryptionPacketBuffer(int(lineData[0]),
int(lineData[1]), lineData[2])
        #Every 800 lines is 1 packet length
        if(lineIdx % LINES_IN_PACKET == 0):
            data.append( packet )
            packet = []


    file.close()

    return data

def SendEncryptionBuffer(timeToSend, bytesToSend, anchorToSend):
    global ANCHORS, currentTime

    timeToSend = round(timeToSend)
```

```
    #Create data
    dataItem = EncryptionPacket()
    dataItem.network_time = timeToSend
    #dataItem.data_payload = byteToSend.ljust(MIN_PACKET_SIZE, ' ')
    #print(('Bytes to send \t: '), bytesToSend)
    dataItem.data_payload = bytes(bytesToSend)

    #Create packet, add header, data payload
    cdp_packet = CDP()
    #This serial number doesn't seem to matter?
    cdp_packet.serial_number = CiholasSerialNumber(0x010403B4)
    cdp_packet.add_data_item(dataItem)
    packet_data = cdp_packet.encode()

    #print(datetime.datetime.now().time(), ' Sending encr packet @ nt: ',
    #    timeToSend, ' currTime', currentTime, '\ndiff time', timeToSend -
currentTime)

    #send packet
    data_socket.sendto(packet_data, ANCHORS[anchorToSend])


args = parser.parse_args()

#Read file
packet_list = ReadEncryptionFile(args.filename)

data_buffer = []

argsK = parserK.parse_args()

#read the key
sha = ReadKeyFile(argsK.filename)

#for item in sha:
#   print(item,  end =" ")
#print('\n')

try:
    while True:#shaIndex < len(sha) + 1: # or awaiting_reply:
        try:
            #Read packets until we get a reply from the receiving anchor
            data, addr = data_socket.recvfrom(65536)
            new_rx_packet = CDP(data) #decode the data into a cdp packet and
decode data items into their appropriate types\

            #print( 'Serial: ', new_rx_packet.serial_number, '\t Addr: ',
addr )

            for data_item in new_rx_packet.data_items_by_type[0x802D]:
                if(not awaiting_reply and shaIndex < len(sha)):
                    if(Ttxlast == 0):
                            Ttxlast = data_item.nt64 + TICKS_PER_SECOND
                            Trxlast = data_item.nt64 + TICKS_PER_SECOND
                            #Ttxlast = data_item.nt64
                            #TrxLast = data_item.nt64
```

```
                    else:
                        slot = sha[shaIndex]
                        shaIndex +=1

                        #print(slot)

                        #Send packet, then wait for reply
                        #TICKS_PER_SECOND is a one frame offset
                        time_to_send = Ttxlast + Tdistlast + TbtwnOffset -
anchorDistances[currentAnchor] + ((slot + 0.5) * Tslot) + TICKS_PER_SECOND

                        #print('Current nt: ', data_item.nt64, 'ShaIndex: ',
shaIndex, 'Scheduled to send: ', int(time_to_send)

                        currentTime = data_item.nt64

                        #packetsToSend = [shaIndex] + packet_list[packetIdx]
if shaIndex > 5 else GARBAGE_DATA
                        packetsToSend = [shaIndex] + packet_list[packetIdx]

                        SendEncryptionBuffer( time_to_send, packetsToSend,
currentAnchor )
                        awaiting_reply = True

                        Ttxlast = time_to_send

                        Tdistlast = anchorDistances[currentAnchor]

                        #nchor += 1
                        if(currentAnchor > 2):
                            currentAnchor = 0

                        keyByte = slot

                        packetIdx += 1

                        if (shaIndex == 33):
                            shaIndex = 0
                            Ttxlast = 0

            #if(addr == ANCHORS[3]):
            if(True):
                if(addr == ANCHOR_01E3_ADDRESS):
                    #print( data_item.nt64)
                    for data_item in
new_rx_packet.data_items_by_type[0x8080]:
                        print('Serial:', new_rx_packet.serial_number, '
Received a packet 0x8080', ' nt: ', data_item.network_time)#, 'shaIdx: ',
data_item.data_payload[0])
                        #print ('payload: ', data_item.data_payload.strip() )
                        #print ('time received: ',
hex(data_item.network_time) )
                        #print('nt: \t\t', data_item.network_time, ' trxLast:
', TrxLast, ' tbtwenoffset: ', TbtwnOffset, 'tslot: ', Tslot)
                        #Calculate keyByte
                        keyByte = (data_item.network_time - TrxLast -
TbtwnOffset - TICKS_PER_SECOND ) / Tslot
```

```
                        #Record shaindex
                        shaIndexR = data_item.data_payload[0]
                        #Remove shaindex from data payload
                        #del data_item.data_payload[0]


                        if(keyByte < 256 and shaIndexR > 0): # and
key[shaIndexR-1] == 0):
                            #key[shaIndexR-1] = round(keyByte)
                            key.append(int(keyByte))

                        #Append payload to buffer (disregard first byte:
shaindex)
                        data_buffer += data_item.data_payload[1:]

                        #Update TrxLast for next recv
                        TrxLast = data_item.network_time

                        print('Key byte = ', hex(int(keyByte)), 'Index: ',
shaIndexR,'\n')#, '\tSha indexR: ', shaIndexR, '\n\n\n')

                        awaiting_reply = False


                        if (shaIndexR == 33):
                            print('THE KEY IS:')
                            for item in key:
                                print(hex(int(item)).lstrip("0x"),  end =" ")
                            print('\n')
                            convertedBuf = convertBuffer(data_buffer)
                            writeToFile(convertedBuf, "Output.txt")
                            convertedKey = convertKeyToHex(key)
                            writeToFile(convertedKey, "key.dat")

                            key.clear()

        except (KeyboardInterrupt, SystemExit):
            data_socket.close()

    print('Done with loop')

    convertedBuf = convertBuffer(data_buffer)
    writeToFile(convertedBuf, "OutputBuffer.txt")
    convertedKey = convertKeyToHex(key)
    writeToFile(convertedKey, "key.dat")

except Exception as e:
    print (e)
```

References

[1] E. F. Brickell, J. H. Moore and M. R. Purtill, “Structure in the S-boxes of the DES, Advances in Cryptology”, presented at Conf. of CRYPTO, United States, 2001.[Online]

[2] Gollman. Dieter, *Computer Security* (2nd ed.). West Sussex, England: John Wiley & Sons, Ltd, 2011, pp. 156-178.

[3] A. Bogdanov, D. Khovratovich, and C. Rechberger. (2012, Jun). Biclique Cryptanalysis of the Full AES. *IEEE Comp. Sci.* [Online] Available: https://www.webcitation.org/68GTcKdoD?url=http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf. [Accessed Nov. 4, 2018]

[4] Hoffman. P, “Cryptographic Suites for IPsec,” IETF, vol. ED-1, pp. 13-29, Dec. 2005.

[5] M. Ciholas, “DWETH111,” *ciholas.com*, para. 2, July. 21, 2017.[Online]. Available: https://www.ciholas.com/product/dweth111/. [Accessed Nov. 28, 2018]