**Cyber Security Encryption and Decryption Project**

*Final Report*

Adeniji Adedamola, Calibuso Leonila Antonette, Cook Billy, Jaimes Edgar, Waldron Dylan

Department of Computer Science and Information Technology

Armstrong State University

April 26th, 2018

Table of Contents

1 Executive summary………………………........................................................................2-3

2 Project background…………….......................................................................................4-8

2.1 Needs statement………………………………………………………………………………....4

2.2 Goal and objectives……………………………………………………………………………4-6

2.3 Design constraints and feasibility……………………………………………………………....7

2.4 Evaluation of alternative solutions…………………………………………………………...7-8

3 Final design…………….....................................................................................................8

3.1 System description……………………………………………………………………………....8

3.2 Approach for design validation………………………………………………………………....8

4 Implementation notes…………….................................................................................9-17

5 Experimental results...................................................................................................18-21

6 User’s Manuals………................................................................................................22-26

7 Appendices…………………………………………………………………………………….26

# 

# **1 Executive summary**

When you log onto your favorite websites you never really think twice about how your passwords are kept safe. When the client and server interact, it needs to be secure. This is done by encrypting the binary “signal” that is sent in packets back and forth. This is a very broad description, but we will get more technical later. The need to encrypt data didn’t start with computers either. It began with written messages and has slowly evolved into the more advanced mathematical formulas we use in our encryption software today. The need for encryption has also transitioned from the few using it to the clear majority. When you fill in medical forms or banking records online you would like that data protected. If you for some reason don’t want it protected, hiding all of your money under your mattress would be a safer bet. So now that the data is encrypted back and forth between the client and server; everyone is safe, protected, and happy. The government also encrypts the data that is stored on their servers, unfortunately, they don’t encrypt it very well as there have been numerous data breaches in the past. This is what makes the encryption and decryption process very important and shows the need for making sure our own files are as secure as they can be.

Our project focuses on the encryption and decryption process as well as displaying it nicely and neatly so the user can understand what they are looking at. The whole purpose of our project is to inform the user of how the encryption and decryption process works and not confuse the user. The code itself may be extremely confusing to people with less coding experience or experience that does not include a solid understanding of Java. However, our project aims at making sure the user understands how exactly the encrypting and decrypting process works, even if the user doesn’t understand the code.

The GUI (Graphic User-Interface) is what helps the user understand what's going on even if they don’t understand what exactly is happening within the code. This is the case of almost all GUI’s and graphics, and ours is no exception. The GUI makes the project very user-friendly so anyone can use it truly understand what we’re trying to show them. The final design will show the user three different graphs with the different data points. One graph for before the encryption, one graph for after the data encryption, and one graph for after the decryption of the encrypted data which essentially reverts it back to its original data. This will be explained more in depth later in the project report and will be made more clear exactly what is being shown in our project.

Our program starts by creating a random data block of 2048 bits and puts that data block into a file to encrypt later when needed. It also gets a random IV and Key which will be used together as well when needed to encrypt our code. TEA (Tiny Encryption Algorithm) is the encryption method we choose to use for our program. Our program takes a binary string of 64 bits and then maps each individual bit in that string into a unique eight-bit decimal value. This encoding technique is based on the randomly generated user’s profile. Our program encrypts the 2048 bits stated before and then displays the encryption of a random 64-bit value. In simple terms, our project takes a binary signal and converts it to random analog signals known as noise, which distorts the original signal data. Even those simple terms may sound confusing, which is why we have a GUI as well as this entire project report to help the user understand whats happening more in-depth.

For people more efficient with coding in Java, our code is available to easily edit and understand with many methods and comments for helping the new user/programmer understand what they’re looking at so they can very easily change the program up or fix anything if it needs to be fixed. Our code is error free and it won’t be necessary to edit any of the code unless the user desires to do so.

The results of our project were exactly as expected, with the decryption and encryption code running perfectly. We even exceeded our original plan for just coding and decrypting by the GUI which wasn’t required and only acts as a bonus for the project. Project management went fairly well, although not perfect. Most of this project requires a good understanding of coding (Java specifically) to do since even the later parts of this report rely heavily on understanding the code, which a large portion of the group doesn’t have. However, we all pitched in by helping with the project report if we didn’t understand the code. We had weekly meetings to make sure we all knew what was going on and what we all needed to do. Not to be biased, but at presentation time, our group definitely had the best looking GUI and most understandable project in the class. The code and report all came together in the end and made for a very nice final result for our project. Over the course of the project and class as a whole, the group learned a lot about crypto which helped lead us to our final result.

# 

# 

# **2 Project background**

## **2.1 Needs statement**

Hardware requirements for the project are any brand of processor with at least a speed of 800 Mhz speed, however, 1.5 GHz or faster is recommended. The minimum free disk space required is 300MB but 1 GB or more is definitely recommended. 512 MB is the required amount of RAM, however, one gigabyte is recommended to run our project efficiently. Running our project on the minimum system requirements if definitely not recommended as it will run slowly, but it will still run either way. Our project is running through the software Eclipse (version 4.7.3 (Oxygen)). We choose Eclipse so the user can see the code that is running as well as the possibility of being able to add a GUI to our project. Using Netbeans IDE 8.2 or a different compiler is also a possibility to run the code, however, all of our testing and work came from using Eclipse so it’s the safest bet. The user will also need to have Java downloaded to be able to run everything properly, with having at least Java 1.4 or higher to run. However, the current version of Java (or at least 5.0) is definitely recommended when running any Java code for any compiler.

## **2.2 Goal and objectives**

This brings us to the goal of our project. The goal of our project is to bring to light one simple way encryption is done using TEA. This is being done for educational purposes as the encryption is much less advanced than any modern encryption scheme used today. The very first and foremost objective we have is that the user gains at least a better understanding of what encryption is and how it works. This goal involves decrypting and encrypting binary values and then displaying them all at the same time in our GUI. While the GUI itself is extra and not essential to our project and main goal, it makes the information given easier to process and understand. The GUI in the case of this project is for extra credit and made the project a little more difficult, but it is a side goal our team had. The GUI’s main feature is the three graphs on the bottom of the screen which show the final encrypted and decrypted data points. These two goals of encrypting and decrypting the data as well as displaying it nicely in our GUI really tie the project together and will help the user understand whats going on in the encryption and decryption process.

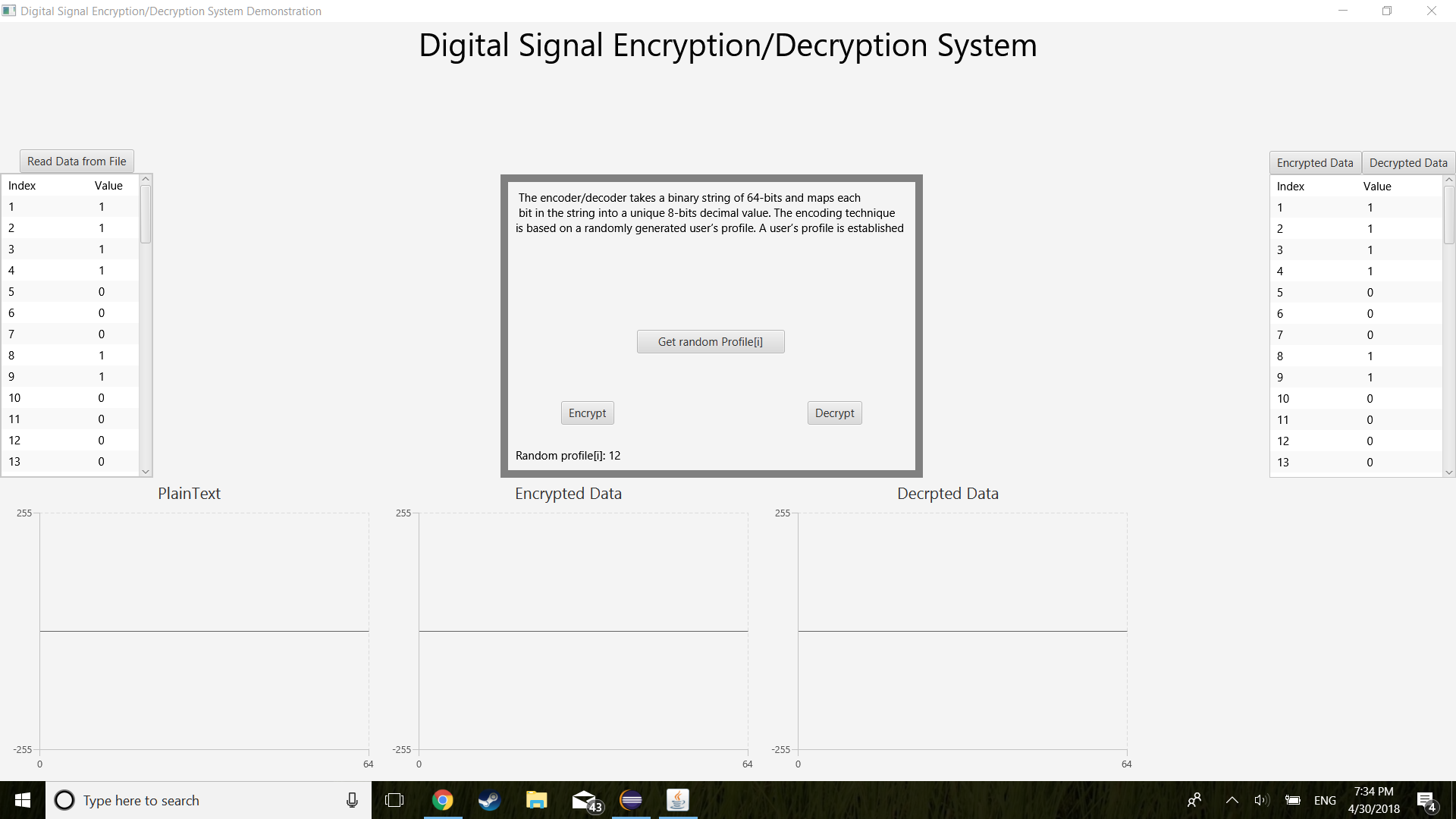


Figure 1: GUI application selecting a profile at random.

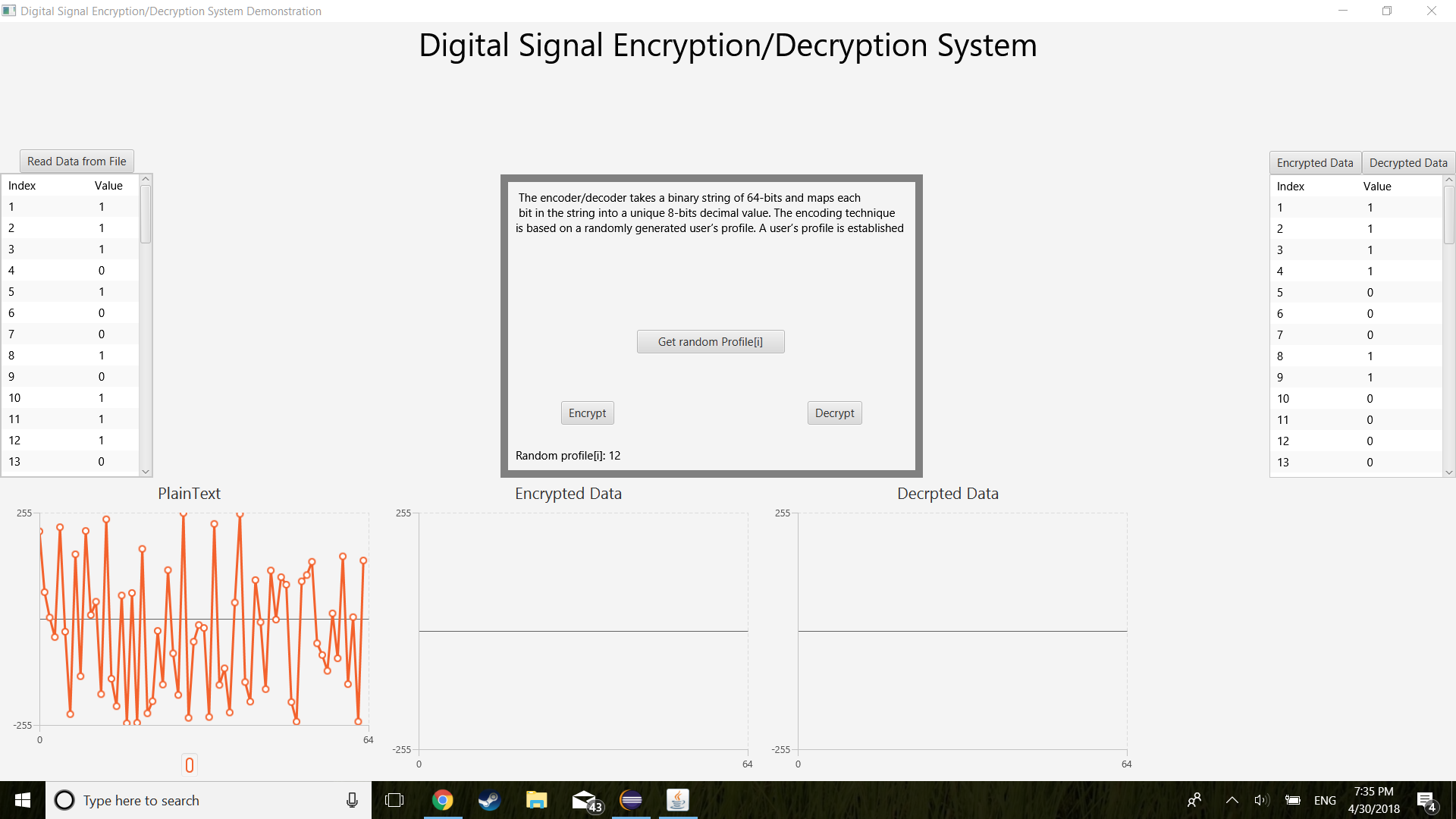


Figure 2: GUI application reading in data from a binary text file.

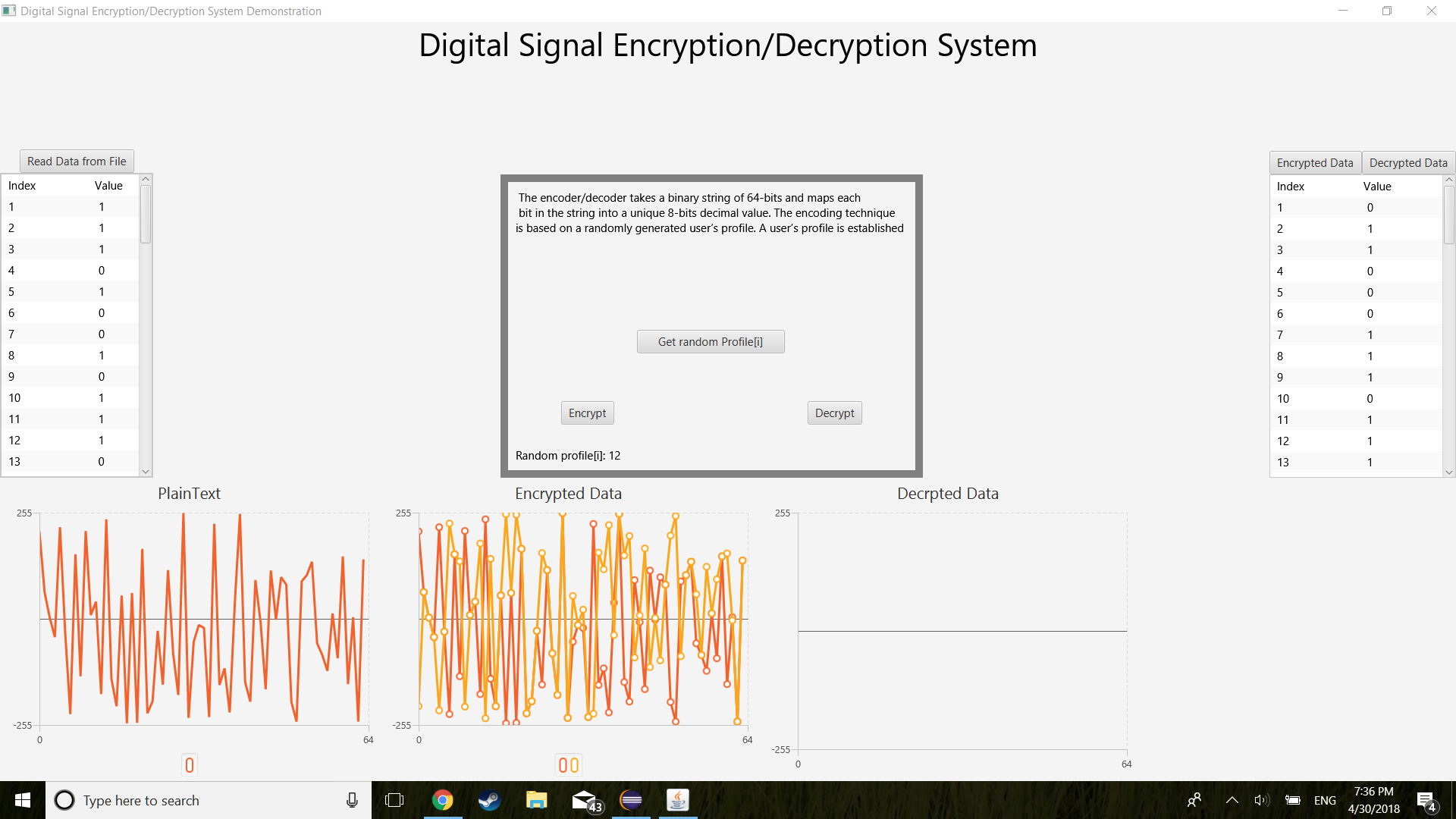


Figure 3: Encryption of the data from the binary text file.

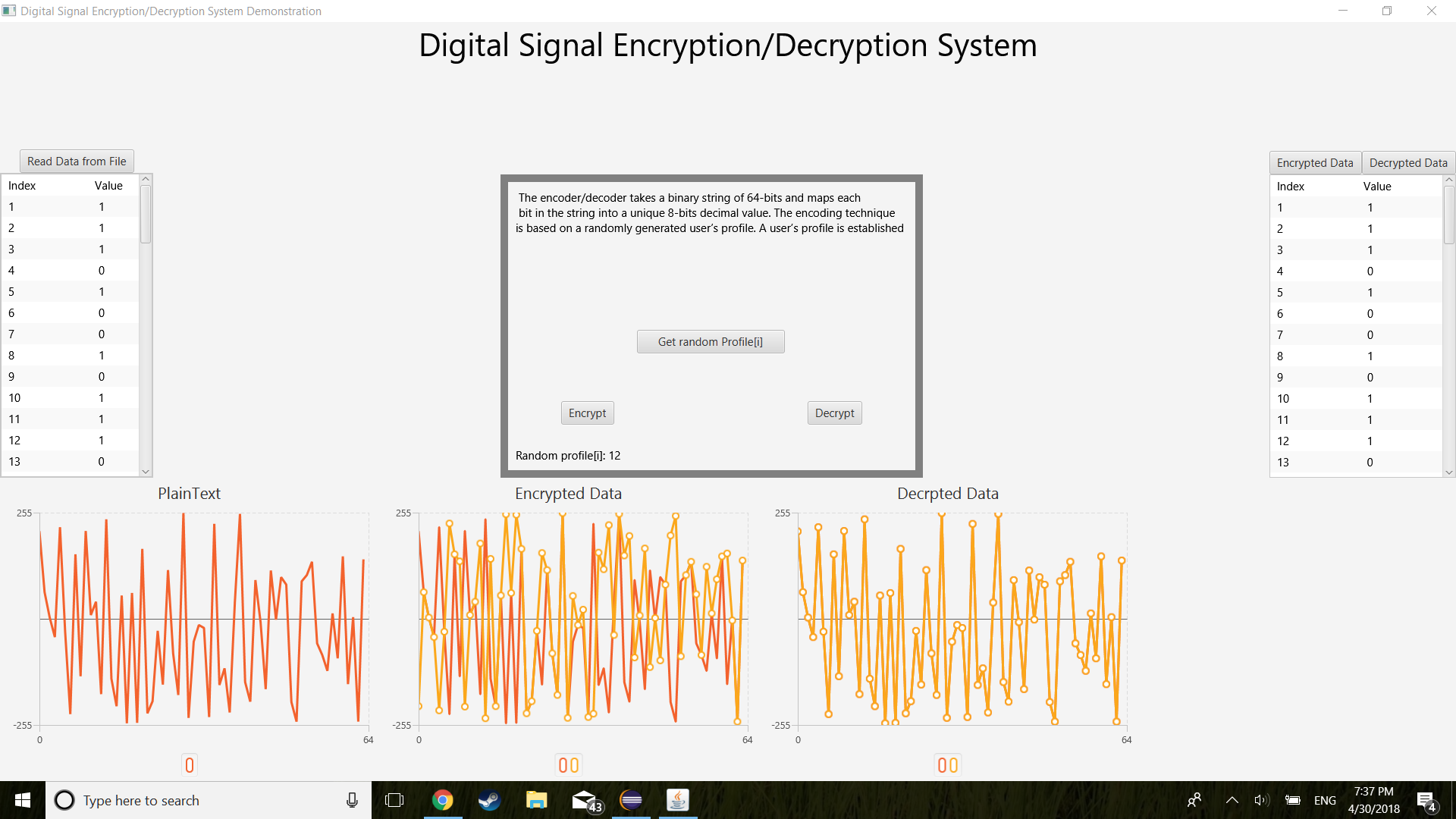


Figure 4: Decryption of the data. Notice the numbers on the left and right of the screen.

## 

## **2.3 Design constraints and feasibility**

The largest technical constraint we have for the project is time. It is feasibly possible to finish on time so the constraint has no real effect on our ability to produce a completed result. The next constraint that we ran into is the amount of programming experience the group has as a whole. The team has been assigned group members with the technical expertise necessary to carry out the project. This caused a lot of confusion as not everyone in the group could understand what was happening within the code. The next constraint is the size of the encryption. The encryption could be so large that our modern laptops couldn’t compile the software without crashing. Fortunately, the project size is not large enough to matter. The last constraint the team ran into was the program used to design the graphical user interface. It was originally going to be designed in Visual Studio, but then the team would have to convert from C# to Java and vice versa. The solution was to just use Java to code and design the entire project itself. The constraints were very mild in this project.

The next constraint that we had to face is the lack of experience. There really was a wide range of practical knowledge in our group. There were some who didn’t understand any bit of code, which of course can actually hinder a programming assignment. A few people understood some different languages of code, but weren’t exactly fluent in Java. Then, there were those in the group who had a great bit of experience in Java and Java GUI building. This type of random mix and match seemed to work for our group but, I don’t think that will always be the case.

## **2.4 Evaluation of alternative solutions**

* One alternate solution was to code this project in a different compiler and with a different language. Some of our group have previously taken courses on Objective C, HTML, and a few others but we all knew doing this project would have been a pain with those languages. A few members of our group had a basic understanding of C# and there was a possibility we could have used Visual Studio for our project instead. However, our group’s experience in Java was better than in C#, considering we only have one C# class offered here and multiple levels of Java available to take. This more in-depth understanding of Java made the choice clear for all of us on which we language we would choose.
* Not having a GUI was also a possibility since it was optional. This would have caused us to lose out on bonus points, but it was a possibility.
* There was a consideration of using a GUI builder for Java. This supposedly would of made the building of the GUI shorter and easier. The application is a plugin for Eclipse and the link is here: <https://www.eclipse.org/windowbuilder/>.
* Another alternate solution was to rip off the example assignment and reword most everything. We didn’t do that, obviously, but, theoretically, it could be a possibility. The con to this would be that it would probably receive a zero and have to deal with having plagiarized.
* The final alternate solution would be to just pay someone to do it for us. Again, we didn’t do that either; it was too expensive. The downfall to this is it would be very unhonest and could risk punishment from the university if caught. Lastly, we really won’t learn anything this way.

# **3 Final design**

## **3.1 System description**

The Digital Signal Encryption and Decryption project is implemented to create an encryption and decryption system that reads a binary signal and converts it to random analog signals. It is composed of one class that makes the entire application run. The DigitalEncrypt\_Decrypt class launches the GUI application that reads the data from a file, creates a random initialization vector and key, and displays the encrypted or decrypted data. For it to run, the event handler function was utilized to execute the encryption and decryption algorithm. The algorithms for both are methods associated with the class, encrypt() and decrypt(). The getSignal() method returns the user profile from an inputted list of integers.

## **3.2 Approach for design validation**

The application encryption and decryption were tested against existing programs of the same manner online. When the outputs were the same then the application was working as designed. If, however, the outputs from our application didn’t match then we knew that our program has an error somewhere in the code. This technique is really no different than using a calculator to check your results. The GUI in our application was made to be extremely handy in checking the tested values. The left of the application shows the original plaintext values, and on the right is the encryption and decryption values. This makes it really easy to debug the program since you can see what the numbers are doing at each step.

# 

# **4 Implementation notes**

The class DigitalEncrypt\_Decrypt.java contains methods to encrypt, decrypt, get signal output, and launch the application. A text document is provided for the string bits: “String.txt”.  
   
  
I. DigitalEncrypt\_Decrypt.java – launches the application; declares variables and array lists; creates encryption, decryption, and signal methods.  
  
 a. Variables  
 i. int size – initializing it to 2048 bits  
  
  
public static int size = 2048;  
  
  
 ii. int blockS – initializing it to 64 bits  
  
  
public static int blockS = 64;  
  
  
 iii. int profile – declared profile variable  
  
  
public static int profile;  
  
  
 b. Arrays  
 i. Integer[] randProfile, encProfile, decProfile – creating a fixed sized array of blockS with 64 bits  
  
  
public static Integer[] randProfile = new Integer[blockS];  
public static Integer[] encProfile = new Integer[blockS];  
public static Integer[] decProfile = new Integer[blockS];  
  
  
 ii. Integer[] IV – creating a fixed sized array of blockS with 64 bits  
  
  
public static Integer[] IV = new Integer[blockS];  
  
  
 iii. Integer[] key – creating a fixed sized array of 128 bits  
  
  
public static Integer[] key = new Integer[128];  
  
  
 c. Array Lists  
 i. ArrayList<Integer[]> encBlock – creates an integer array list of encryption block  
  
  
public static ArrayList<Integer[]> encBlock = new ArrayList<Integer[]>();  
  
  
 ii. ArrayList<Integer[]> tempEnc – creates an integer array list of a temporary encryption block  
  
  
public static ArrayList<Integer[]> encBlock = new ArrayList<Integer[]>();  
  
  
 iii. ArrayList<Integer[]> tempDec – creates an integer array list of a temporary decryption block  
  
  
public static ArrayList<Integer[]> tempDec = new ArrayList<Integer[]>();  
  
  
 iv. ArrayList<Integer[]> plainBlock – creates an integer array list of a simple plain block  
  
  
ArrayList<Integer[]> plainBlock = new ArrayList<Integer[]>();  
  
  
 v. ArrayList<Integer[]> tempBlock – creates an integer array list of a temporary block  
  
  
ArrayList<Integer[]> tempBlock = new ArrayList<Integer[]>(plainBlock);  
  
  
 vi. ArrayList<Integer[]> tempB2 – creates an integer array list of a second temporary block  
  
  
ArrayList<Integer[]> tempB2 = new ArrayList<Integer[]>(plainBlock);  
  
  
 vii. ArrayList<Integer[]> decBlock – creates an integer array list of encryption block  
  
  
ArrayList<Integer[]> decBlock = new ArrayList<Integer[]>(encrypt(tempB2));  
  
  
 d. Methods  
 i. start(Stage primaryStage) throws IOException – contains code for GUI  
  
 ii. handle(ActionEvent e) – is used to get random block of profile, to read the plaintext, encryption and decryption  
  
  
// Get random block!!!!---------------------------------------------------------  
 Button getProfile = new Button("Get random Profile[i]");  
 BorderPane.setAlignment(getProfile, Pos.CENTER\_LEFT);  
 getProfile.setPrefWidth(400);  
 encDec.setCenter(getProfile);  
 getProfile.setOnAction(new EventHandler<ActionEvent>() {  
 @Override  
 public void handle(ActionEvent e) {  
 profile = (int) (Math.random() \* 32);  
 Text prof = new Text("Random profile[i]: " + Integer.toString(profile));  
 BorderPane.setAlignment(prof, Pos.BOTTOM\_LEFT);  
 encDec.setBottom(prof);  
 plainChart.getData().clear();  
 encryptChart.getData().clear();  
 decryptChart.getData().clear();  
   
 }  
 });  
  
  
// EVENT HANDLER read plaintext --------------------------------------------  
 readData.setOnAction(new EventHandler<ActionEvent>() {  
 @SuppressWarnings({ "unchecked" })  
 @Override  
 public void handle(ActionEvent e) {  
 plainChart.getData().clear();  
 randProfile = plainBlock.get(profile);  
 for (int i = 1; i < 65; i++) {  
 if (i < 10)  
 bits[i].setText(i + " " + Integer.toString(randProfile[i - 1]));  
 else  
 bits[i].setText(i + " " + Integer.toString(randProfile[i - 1]));  
 }  
 Integer[] pData = getSignal(randProfile);  
 for(int j = 0; j < blockS; j++) {  
 plainLine.getData().add(new XYChart.Data(j, pData[j]));  
 }  
 plainChart.getData().add(plainLine);  
 }  
 });  
  
  
// Encryption Event Handler---&&---read data button--------------  
 XYChart.Series tempLine = plainLine;  
  
 encrypt.setOnAction(new EventHandler<ActionEvent>() {  
 @SuppressWarnings({ "unchecked", "rawtypes" })  
 @Override  
 public void handle(ActionEvent e) {  
 tempEnc = encrypt(tempBlock);  
 encProfile = tempEnc.get(profile);  
  
 encBits[0] = new Text("Index Value");  
 for (int i = 1; i < 65; i++) {  
 if (i < 10)  
 encBits[i].setText(i + " " + Integer.toString(encProfile[i - 1]));  
 else  
 encBits[i].setText(i + " " + Integer.toString(encProfile[i - 1]));  
 }  
 Integer[] eData = getSignal(encProfile);  
 for(int j = 0; j < blockS; j++) {  
 encLine.getData().add(new XYChart.Data(j, eData[j]));  
 }  
 encryptChart.getData().addAll(tempLine, encLine);  
 }  
 });  
  
  
//ENCRYPTED DATA VIEW BUTTON------------------------------------  
 encData.setOnAction(new EventHandler<ActionEvent>() {  
   
 @Override  
 public void handle(ActionEvent e) {  
   
 encBits[0] = new Text("Index Value");  
 for (int i = 1; i < 65; i++) {  
 if (i < 10)  
 encBits[i].setText(i + " " + Integer.toString(encProfile[i - 1]));  
 else  
 encBits[i].setText(i + " " + Integer.toString(encProfile[i - 1]));  
 System.out.print(Integer.toString(encProfile[i - 1]) + " ");  
 }  
 }  
 });  
  
  
//DECRYPTION BUTTON----------------------------------------------------  
 Decrypt.setOnAction(new EventHandler<ActionEvent>() {  
 @SuppressWarnings({ "unchecked" })  
 @Override  
 public void handle(ActionEvent e) {  
 tempDec = decrypt(decBlock);  
  
 decProfile = tempDec.get(profile);  
 encBits[0] = new Text("Index Value");  
 for (int i = 1; i < 65; i++) {  
 if (i < 10)  
 encBits[i].setText(i + " " + Integer.toString(decProfile[i - 1]));  
 else  
 encBits[i].setText(i + " " + Integer.toString(decProfile[i - 1]));  
 }  
 Integer[] dData = getSignal(decProfile);  
 for(int j = 0; j < blockS; j++) {  
 decLine.getData().add(new XYChart.Data(j, dData[j]));  
 }  
 decryptChart.getData().addAll(tempLine, decLine);  
 }  
 });  
  
  
//DECRYPT DATA VIEW BUTTON--------------------------------------------  
 decrData.setOnAction(new EventHandler<ActionEvent>() {  
 @Override  
 public void handle(ActionEvent e) {  
   
 encBits[0] = new Text("Index Value");  
 for (int i = 1; i < 65; i++) {  
 if (i < 10)  
 encBits[i].setText(i + " " + Integer.toString(decProfile[i - 1]));  
 else  
 encBits[i].setText(i + " " + Integer.toString(decProfile[i - 1]));  
 System.out.print(Integer.toString(decProfile[i - 1]) + " ");  
 }  
 }  
 });  
  
 iii. Main(String[] args) – launches the program  
  
 iv. encrypt(ArrayList<Integer[]> list) – uses two loops within a loop to apply Tiny Encryption Algorithm (TEA). Each integer in the list with the size of blockS gets XOR with the key and IV adding i and performs a circular shift to the left.  
   
  
public static ArrayList<Integer[]> encrypt(ArrayList<Integer[]> list) {  
 Integer[] shiftedK = key;  
 Integer[] newIV = IV;  
 ArrayList<Integer[]> encList = new ArrayList<Integer[]>();  
   
 // TEA--------- Run EX0r through IV and 128 bit Key-------------------------  
 for (int i = 0; i < list.size(); i++) {  
 Integer[] temp = list.get(i);  
 for (int j = 0; j < blockS; j++) {  
 temp[j] = temp[j] ^ newIV[j];  
 temp[j] = temp[j] ^ shiftedK[j];  
 }  
 encList.add(temp);  
 // +ADD+ i TO IV--------------------------------------  
 String n = Integer.toBinaryString(i);  
 for (int k = 1, s = 0; s < n.length(); s++) {  
 newIV[blockS - k] = newIV[blockS - k] ^ n.charAt((n.length() - 1) - s);  
 k++;  
 }  
 // SHIFT KEY TO THE LEFT EVERY BLOCK---------------------------  
 for (int p = 0; p < 5; p++) {  
 for (int t = 0; t > 127; t++) {  
 shiftedK[t] = shiftedK[t - 1];  
 shiftedK[0] = shiftedK[shiftedK.length - 1];  
 }  
 }  
 }  
 return encList;  
 }  
   
  
 v. decrypt(ArrayList<Integer[]> list) – uses for loops to run TEA decryption algorithm; the decryption process is the inverse of the encryption function. This will return the list into its original position using the IV and the shiftedK variables.  
   
  
public static ArrayList<Integer[]> decrypt(ArrayList<Integer[]> list) {  
 Integer[] shiftedK = key;  
 Integer[] newIV = IV;  
 ArrayList<Integer[]> decList = new ArrayList<Integer[]>();  
 Integer[] c = new Integer[blockS];  
  
 // TEA--------- Run EX0r through IV and 128 bit Key-------------------------  
 for (int i = 0; i < list.size(); i++) {  
 Integer[] temp = list.get(i);  
 for (int j = 0; j < blockS; j++) {  
 temp[j] = temp[j] ^ newIV[j];  
 temp[j] = temp[j] ^ shiftedK[j];  
 }  
 decList.add(temp);  
  
 // +ADD+ i TO IV-----------------------------------  
 String n = Integer.toBinaryString(i);  
 for (int k = 1, s = 0; s < n.length(); s++) {  
 newIV[blockS - k] = newIV[blockS - k] ^ n.charAt((n.length() - 1) - s);  
 k++;  
 }  
 // SHIFT KEY TO THE RIGHT EVERY BLOCK---------------------------  
 for (int p = 0; p < 5; p++) {  
 for (int t = 0; t > 127; t++) {  
 shiftedK[t] = shiftedK[t - 1];  
 shiftedK[0] = shiftedK[shiftedK.length - 1];  
 }  
 }  
 }  
 return decList;  
 }  
  
   
 vi. getSignal(Integer[] list) – returns profile from initialized profile array of integers. For each list, if the current list is equal to 0, the negative value of the profile is returned.  
   
public static Integer[] getSignal(Integer[] list) {  
 Integer profile[] = { 210, 64, 3, 44, 105, 31, 255, 155, 80, 211, 9, 41, 181, 100, 144, 210, 56, 5, 62, 250,  
 168, 78, 198, 29, 158, 117, 83, 183, 254, 100, 55, 15, 22, 236, 145, 159, 119, 225, 39, 252, 152, 199,  
 93, 8, 169, 116, 2, 100, 82, 200, 247, 90, 105, 137, 59, 87, 125, 13, 95, 150, 157, 4, 247, 140 };  
 Integer[] currL = list;  
 for (int i = 0; i < list.length; i++) {  
 if (currL[i] == 0)  
 profile[i] = -1 \* profile[i];  
 }  
 return profile;  
 }

# 

# 

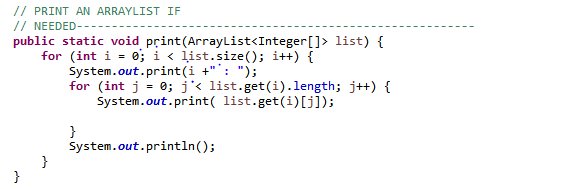
# 

# 

# **5 Experimental results**

The print function was used to validate the Encryption and Decryption functions. Before the GUI was made the print function was essential to correct the code.

1. Print function for large data.



1. Thirty two blocks of 64 bits in order to search for block of 64 bits.

// Create random 2048 bits and input into file----

for (int i = 0; i < size; i++) {

code[i] = (int) (Math.random() \* 2);

output.print(code[i]);

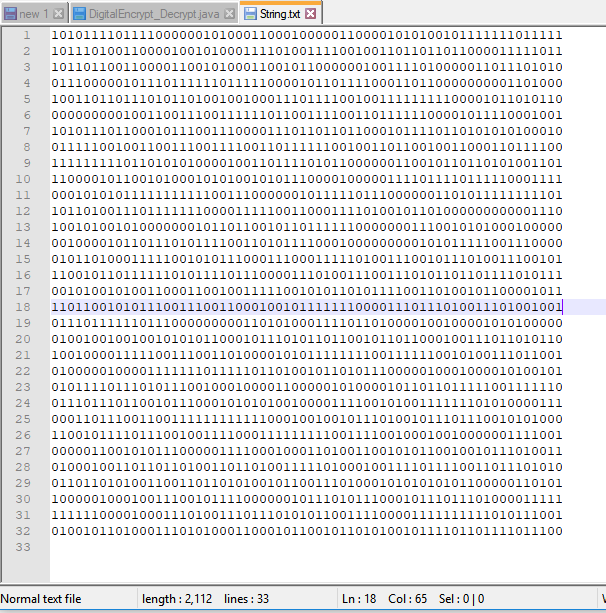
if ((i + 1) % 64 == 0) {

output.println();

}

}

output.close();

1. Print profile block in GUI helps search for the block index.

// Get random block!!!!-----------------------------------

Button getProfile = new Button("Get random Profile[i]");

BorderPane.setAlignment(getProfile, Pos.CENTER\_LEFT);

getProfile.setPrefWidth(400);

encDec.setCenter(getProfile);

getProfile.setOnAction(new EventHandler<ActionEvent>() {

@Override

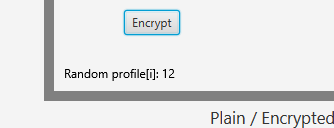
public void handle(ActionEvent e) {

profile = (int) (Math.random() \* 32);

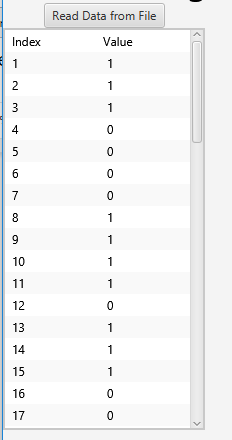
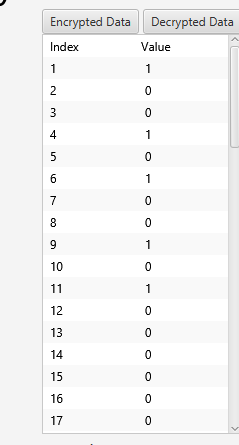
Text prof = new Text("Random profile[i]: " + Integer.toString(profile));

BorderPane.setAlignment(prof, Pos.BOTTOM\_LEFT);

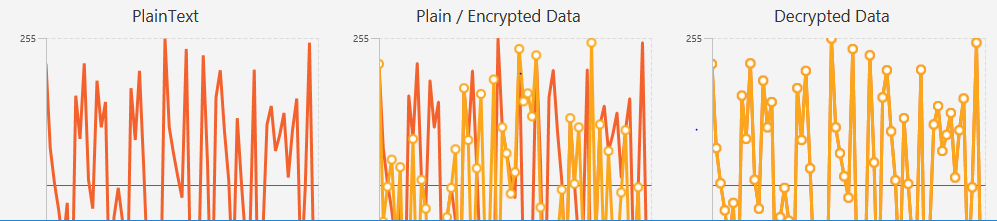
encDec.setBottom(prof);



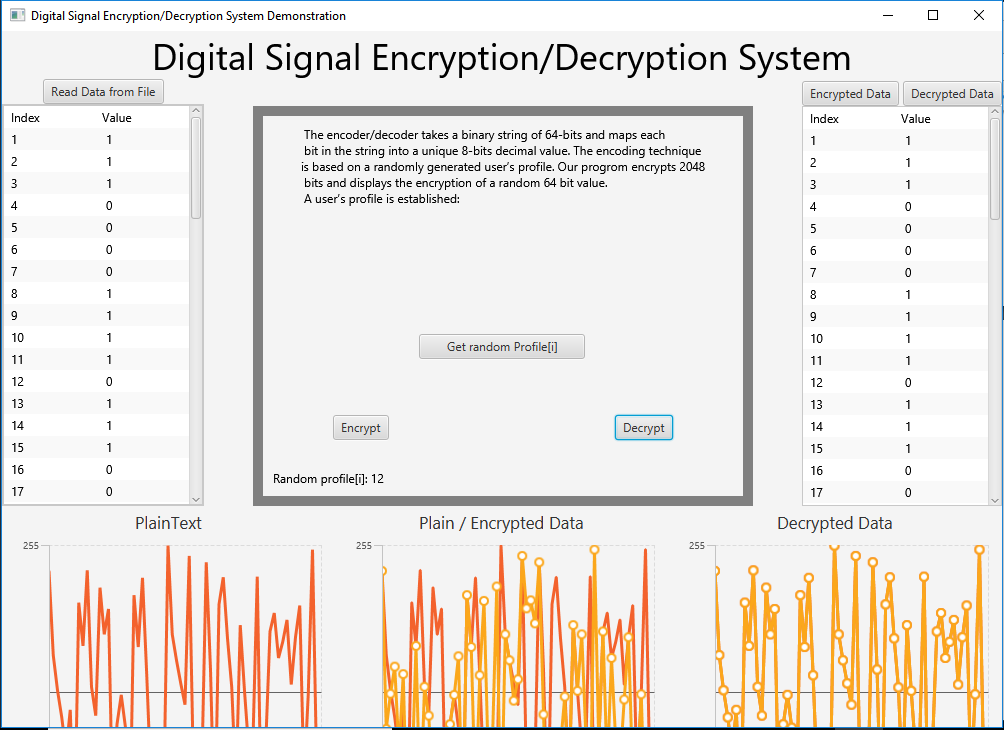
4. Once the index is found on print out. A comparison can start by looking on the GUI, then look at the printed list to compare the values in the profile index.



5. Graphs. Used to compare profiles by plotting the plain 64 bit block on the first graph, the plain block and the encrypted block in the middle graph, then the plain block and the decrypted on 3.



**CTR with 2048 bits of data being encrypted and decrypted while viewing only one block of 64 bits. Gui with plain data, encrypted data, decrypted data, and all graphs.**

****

# 

# **6 User’s Manuals**

The only program needed to run the GUI application is Eclipse. The system requirements to install Eclipse (version 4.7.3 (Oxygen)) are as follows:

* Java version 1.4.0 or greater
* Memory 512MB or greater
* Free disk space of 300MB or greater
* Processor speed 800 Mhz or greater
* Administrative privileges on your system

To install Eclipse please follow this link: <http://www.eclipse.org/downloads/>. There you will find install instructions and the download link for Eclipse.

Now, once Eclipse has been installed go ahead and double click on the executable file.

The first screen to pop up should look like Figure 6.1 below:



Figure 6.1: GUI starting screen on startup.

Once you have the screen in Figure 6.1 we can move on to getting your random profile. To get your random profile please press the button labeled ‘Get random profile[i]. Once you have done that your randomly generated profile number will be displayed. This is all displayed in Figure 6.2 (The number displayed in the random profile may differ than the one in the figure).



Figure 6.2: Random profile selection

After the generation of a profile, the user can select to read in the data that is stored in the file. To do this the user must press the button in the upper right corner labeled ‘Read Data from File’. The button to press is displayed in figure 6.3.



Figure 6.3: Button to read in the data from the file.

When you press the ‘Read data from file’ button the plaintext graph will populate. In figure 6.4 you can see an example of the plaintext being populated.

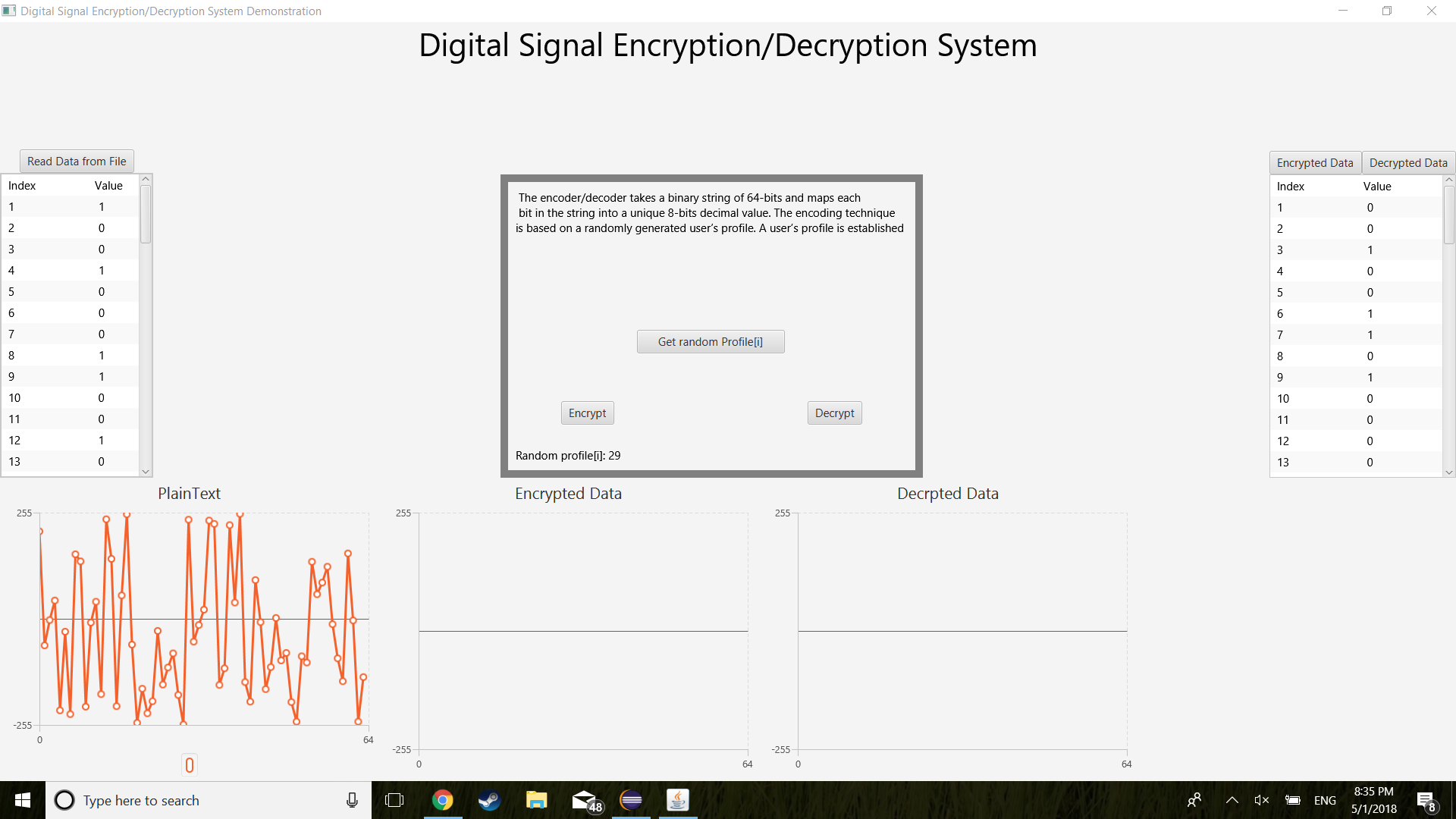


Figure 6.4: Plaintext graph being populated from file.

The next step would be to encrypt the plaintext binary data. To do this you would press the ‘Encrypt’ button. This will then show the encrypted data in the encrypted data graph located in the bottom middle of the application. If you also look to the right-hand side of the graph you can see what the encrypted data is. Figure 6.5 shows the button location and what the encrypted data graph should look like.



Figure 6.5: Graph showing the encrypted data.

The final part of the application is to decrypt the data. This can be done by pressing the ‘decrypt’ button. After pressing the button the decrypted data graph will populate with the decrypted data. The index numbers on the left and right will also be the same after the decrypt button is pressed. Figure 6.6 shows the location of the decrypt button and what a populated decryption graph would look like.



Figure 6.6: Decrypt button location and populated graph of decrypted data.

# **7 Appendices**

Burnette, Ed. “Eclipse IDE Pocket Guide.” *O'Reilly | Safari*, O'Reilly Media, Inc., [www.safaribooksonline.com/library/view/eclipse-ide-pocket/0596100655/ch01s03.html](http://www.safaribooksonline.com/library/view/eclipse-ide-pocket/0596100655/ch01s03.html).