A Time-Efficient Symmetric Key Encryption Technique for ECG Waveforms

Pooja Premnath[1], R. Prahalad[2], Sanjai Balajee[3], Chamundeswari A[4], Dabbu Suman Reddy[5]

[1,2,3,4] Department of Computer Science and Engineering,

Sri Sivasubramaniya Nadar College of Engineering, Chennai

[5] Department of Biomedical Engineering, University College of Engineering,

Osmania University, Hyderabad, Telangana

pooja2110152@ssn.edu.in;
prahalad2110443@ssn.edu.in;
sanjai2110173@ssn.edu.in;
chamundeswaria@ssn.edu.in; dabbu_suman@osmania.ac.in

Abstract

The transmission of medical data, like Electrocardiograms (ECGs), has to be done in a systematic and secure manner. Encryption techniques, especially symmetric methods, using a single key, are simple methods to encrypt data. This paper deals with a novel method for the encryption of ECG waveforms, in the form of .wav files, using matrices. Typical methods that make use of matrices are computationally intensive, especially as the size of the file increases. Unlike other methods, this paper focuses on enhancing the overall computation process, for both encryption and decryption. The proposed algorithm drastically reduces the overall time taken, by arriving at the ideal dimensions of the matrix, to minimize the padding of zeroes, in order to make it square. This reduces the overall difficulty of generating a random key, as well as finding its inverse for decryption. Metrics are also applied to evaluate the overall performance of the algorithm.

Keywords: Electrocardiogram (ECG), Signal Encryption, Signal Decryption, Symmetric Key Encryption

Introduction

Medical data is highly sensitive, containing private patient records and information on their health insurance. This kind of data must be encrypted, to ensure patient confidentiality and protect it from the risk of being manipulated. Transmission of such data for remote analysis must also be done in a secure fashion. Health providers that do not make encryption of their patient data a priority are at risk of hefty penalties and lawsuits, and more importantly, a loss of trust of their patients. The Electrocardiogram (ECG) is an important signal, that may require to be transmitted from one healthcare center to another. In this paper, we propose a method to encrypt and decrypt ECG signals, using a simple private key encryption technique, considering the computational complexity. The performance of the proposed algorithm is also analyzed using parameters to measure computation time and techniques to measure the similarity between the encrypted and decrypted waveforms.

An Electrocardiogram (ECG or EKG) is a tool used to record electrical signals from the heart. This is used to check any abnormalities in the heartbeat, and underlying heart conditions. An ECG is developed by a nerve impulse stimuli to the heart, through electrodes. These electrodes sense the electrical activity of the heart, after placing them on different parts of the chest, and on the skin. The components of an ECG include the P wave, a T wave, and a QRS complex (Ojha and Subashini)[8]. The conventional ECG setup has 12 leads, which are of two kinds-limb leads and precordial leads. The limb leads monitor the heart along the vertical plane, while the precordial leads take readings along the horizontal plane (Sattar and Chhabra)[9]. ECG signals are not periodic in nature. The different waves, P, T, and the QRS complex have different frequencies. The QRS has a high frequency of oscillation. The T region has lower frequencies, and the P region has even lower frequencies (Algarni et al.)[1].

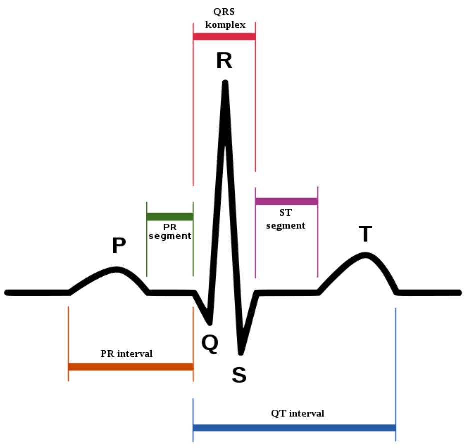

Figure 1 (Cresswell et al.)[4]: An ECG Waveform and its segments and intervals

Existing methods used for encrypting .wav files involve capturing the size of the original wave file, and increasing its size until it is a square number. Zeroes are padded, to make the entire matrix square, and then another random matrix of the same size is generated, that acts as the private key (Al-laham et al.)[2]. In the case of stereo wave files, with two channels, the sampled data from one channel is concatenated at the end of the other channel, and then the column matrix, hence generated is made a square (Nadir et al.)[7]. Another method of encryption proposed to be used on ECG data involves the fusion of wavelet coefficients with a random signal. A chaos-based encryption algorithm is also developed, wherein a combination of fusion, substitution, and chaotic permutations are used. This method of encryption features multiple processes and hence makes it much more secure (Algarni et al.)[1]. Symmetric block ciphers like the AES algorithm have also been applied to ECG data, by means of performing successive contrasting transformations on data block bits iteratively. Though quite secure, the AES algorithm is found to have very high computational complexity (Hameed et al.)[5]. Another system developed involves the encryption of vital signs, right at a Body Area Network (BAN), while the ECG is conducted. This method takes into consideration the fact that BANs utilize very low power, and hence complex encryption schemes like AES or ellipse curve cryptography would lead to high power consumption. The different parameters of the ECG signal are taken as parameters, to generate the initial keys. The encryption is done by generating a stream cipher, using the linear feedback shift register (LFSR). This technique facilitates fast conversion speeds, along with the usage of simple hardware (Bai et al.)[3].

In techniques where matrix operations are used on the sampled data, the column matrix has to be made square, for any kind of arithmetic operations to be performed. In this scenario, zeros are padded, to all of the remaining columns, making the calculation of the product of the original sampled data and the random key, along with the inverse of the matrix, for decryption, computationally very complex. Rather, if the sampled data is optimized in such a fashion, that the least number of zeroes have to be added to either rows or columns, to generate a square matrix, the process would be much more efficient, and less time-consuming.

The ECG waveforms that are analyzed, encrypted, and decrypted in this paper are from the MIT-BIH Arrhythmia Database, which contains ambulatory ECG recordings, each for around thirty minutes (Moody and RG)[6]. The proposed method uses an ECG waveform in a .wav file format. The Waveform Audio File Format, stores audio in the form of bitstreams, as chunks. The .wav file is a representation of samples of sound waves, that vary from the equilibrium pressure of the air (Upadhyay)[10]. The sampled data of the ECG is read, along with the sampling rate from the .wav file. This leads to almost 1300000 values of sampled values, in a concatenated column matrix, obtained from the two-channel data. The generation of a square matrix with zeros padded to the remaining columns becomes very difficult. In cases where the ECG has twelve leads, the length of a single concatenated column would become even larger. In this paper, we propose a methodology to efficiently carry out matrix operations for private key encryption. An algorithm is developed to reshape the column matrix into another matrix, such that the number of zeroes that have to be added to the rows and columns to make a square matrix is minimal. A random key matrix with the size of the square matrix is generated and multiplied with the original data to produce an encrypted product. In order to decrypt the matrix, the inverse of the key matrix is computed, the padded zeroes are removed, and the matrix is then reshaped to its original two-channel format.

The paper initially presents a block diagram describing the methodology proposed. The waveform is first read, and the unencrypted data is plotted and analyzed. The next section presents the flow of the program used to implement the methodology, along with the pseudocode. The encrypted waveform is then plotted, and analyzed with respect to the original waveform. The following section describes the process of decryption and compares the decrypted values with the original matrix. Most importantly, the variation in computation time is analyzed, in the scenarios where the number of zeroes padded is not optimized versus the time taken using the proposed algorithm. The level of security of the solution and the possibilities of discovering the secret key are discussed. Finally, the results of the simulation are discussed using metrics like the Structural Similarity Index (SSM), and other quality metrics like Log Likelihood Ratio (LLR) and the Signal-to-Noise-Ratio (SNR). Histograms of the encrypted and decrypted signals are also plotted to visualize the distribution of the signals.

The Proposed Symmetric Key Technique on ECG Data

The following block diagram shows the process of encrypting and decrypting the ECG waveform.

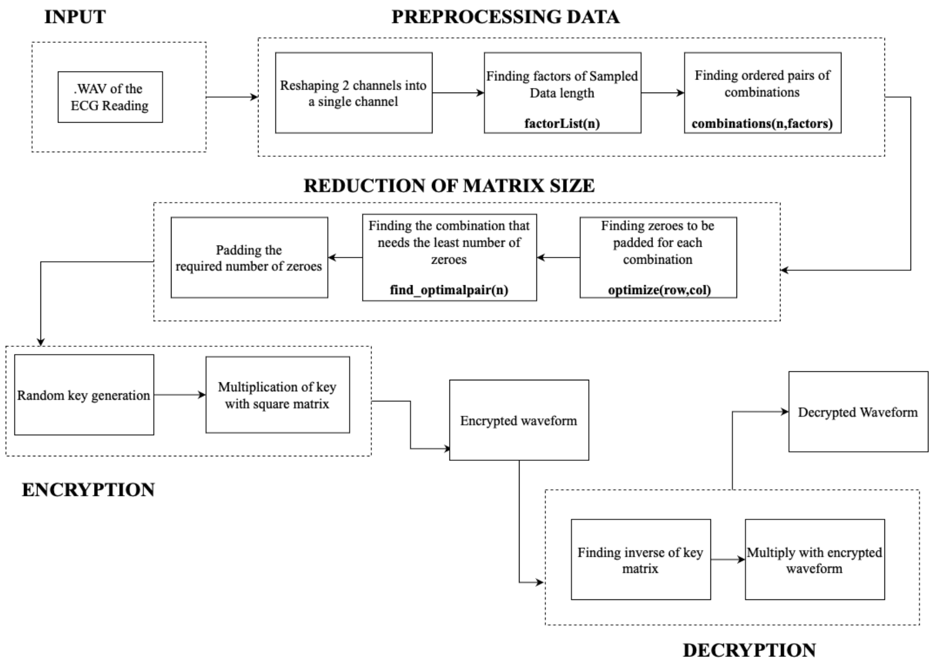
INPUT
.WAV of the ECG Reading
PREPROCESSING DATA
Reshaping 2 channels into a single channel
Finding factors of Sampled Data length
factorList(n)
Finding ordered pairs of combinations
combinations(n,factors)
REDUCTION OF MATRIX SIZE
Finding zeroes to be padded for each combination
optimize(row,col)
Finding the combination that needs the least number of zeroes
find_optimalpair(n)
Padding the required number of zeroes
Random key generation
Multiplication of key with square matrix
ENCRYPTION
Encrypted waveform
Decrypted Waveform
Finding inverse of key matrix
Multiply with encrypted waveform
DECRYPTION

Figure 2: Block diagram of the processing, optimization, encryption, and decryption of ECG Data

In order to carry out the proposed methodology, MATLAB is used to read and perform matrix operations, and Python, along with the libraries, Numpy, Pandas, and Matplotlib are used for data visualization and analysis.

Processing the ECG Waveform

Initially, the .wav file is read, and the sample data, along with the sampling rate is obtained. The .wav file has two-channel ECG data. The first ten seconds of the original waveform, with both channels, are shown below in Figure 3.

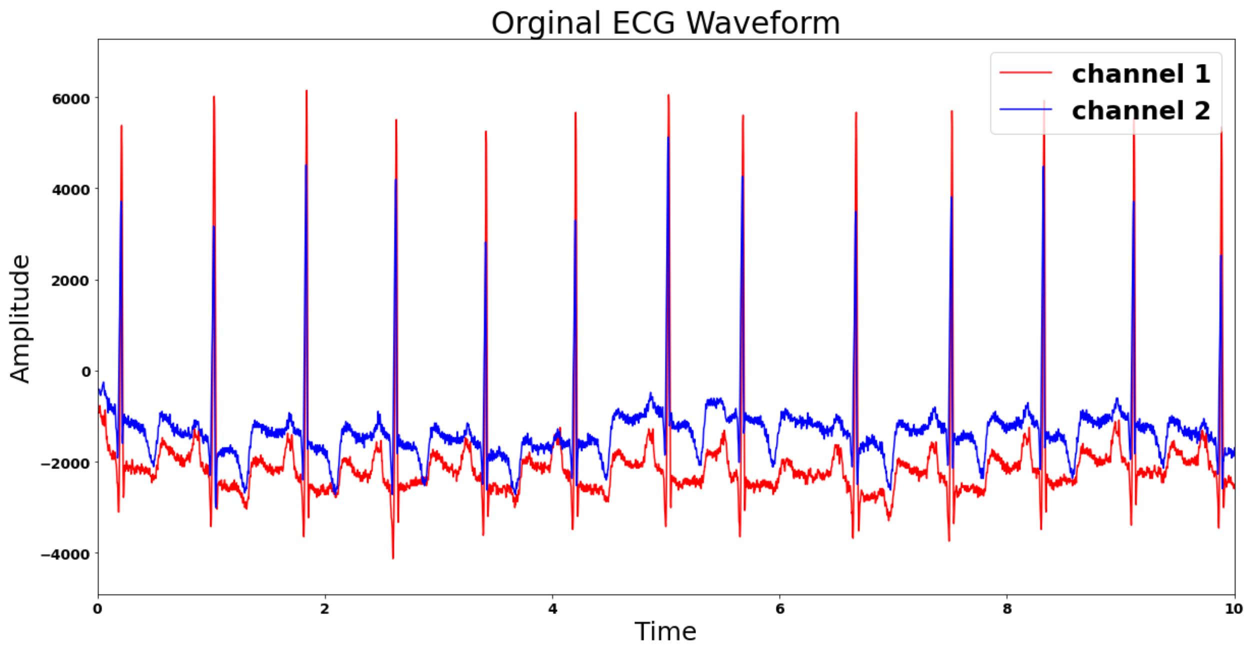

Figure 3: Ten seconds of the original, unencrypted ECG Waveform

The two-channel waveform is concatenated, to obtain a column matrix. The proposed methodology is divided into three parts: making an optimally padded square matrix, encryption

Encryption of the ECG Waveform

In order to encrypt the ECG waveform, the column matrix must be converted into a square matrix, in order to perform matrix operations, such as the computation of the inverse. The square matrix is generated by padding zeroes, to the column matrix. However, if there are *n* rows in the sampled data of the column matrix, *n-1* columns of zeroes would have to be added. In the patient record chosen for encryption, there are 1300000 rows of sampled data, meaning that 1299999 columns of zeroes would have to be padded. This operation is highly inefficient and makes computing the inverse of a 1300000 x 1300000 very expensive and time-consuming. In order to decrease the number of zeroes that are padded, it is proposed that the column matrix first be rearranged into an $m \times n$ matrix, with m rows and n columns. If r is the total number of rows in the column matrix, then that value of m and n is chosen such that $m \times n = r$, and the difference between the smallest square matrix that can be made from the chosen dimensions and $m \times n$ is minimum. This would minimize the number of zeroes that have to be padded.

Pseudocode for Optimization of Zero Padding

The program to compute the configuration of rows and columns in the matrix, that would lead to the least number of zeroes being padded is written using three user-defined functions. Initially, a function *factorList(n)* computes all the factors of the total length of the column matrix. Next, the function *combination(n, factors)*, finds possible combinations such that the product of the ordered pair is equal to the number of rows in the column matrix. Then, the function *optimize(row, col)* is used to find out the side length of the smallest square matrix, that can be made from every ordered pair created. The number of zeroes that would have to be padded in each case is computed. For every set of ordered pairs created, the number of zeroes to be padded is calculated in the function *find_optimalpair(n)*, and the pair which requires the least number of zeroes to be added is called the optimal pair. The column matrix is then accordingly reshaped in the dimensions of the optimal pair.

Pseudocode

Function to find the factors of the sampled data length

factorList(n)

```
Input: n
Output: Array containing the factors of n
Variables: Integer i, n, count
           Array f_list
  1. i ← 1
  2. count ← 0
  3. for i=1 to n//2
     {
         if n%i=0{
                     F_list[count]  ← i
                     count ← count+1
         }
     }
  4.  return f_list
```

Function to generate ordered pairs from the factors

combinations(n,factors)

```
Input:n, factors
Output: Array pairs
Variables: Integer n,i,element,j,count
          Array pairs          //An array to store all possible ordered pairs of factors
           Array factors   //An array containing  factors of  the number of rows in the matrix

  1. count ←0
  2. for  i=0 to factors_length
     {
              element ← factors[i]
              for j=0 to factors_length
              {
                     if element*factors[j]=n
```

```
                    {
                        if (factors[j],elements) ∉ pairs
                        {
                            pairs[count] ← (factors[j],elements)
                            count ← count+1
                        }
                    }
                }
            }
    3.  return  pairs
```

Function to compute the smallest sized square matrix for a given row and column dimension, and thereby the number of zeros to be padded.

optimize(row,col)

Input*: row, col*
Output: *Integer zero_pad*
Variables: *Integer row, col,*

```
        Integer zero_pad            //Total number of zeroes to pad to make a square matrix
        Integer square_size         //The side of the smallest square, from the given dimensions

    1.  if col>=row
        {
            square_size ← col
        }
        else
        {
            square_size ← row
        }
    2.  zero_pad ← (square_size*square_size)-(row*col)
    3.  return  zero_pad
```

Function to find the ordered pair that would require the least number of zeroes to be padded

find_optimalpair(n)

Input*:n*
Output: *Array optimal_pair*
Variables: *Integer n, min_zero, i, zeroes, factors_length*

```
        Array  pairs, f_list, optimal_pair

    1.  f_list ← factorList(n)
    2.  pairs ← combinations(n,f_list)
    3.  min_zero ← None
    4.  optimal_pair ← pairs[0]
    5.  for  i=0 to factors_length
        {
            zeroes ← optimize(pairs[i][0],pairs[i][1])
```

```
          if min_zero=None
          {
                    min_zero ← zeroes
          }
          else
          {
                    if min_zero < zeroes
                    {
                              min_zero ← zeroes
                              optimal_pair ← pairs[i]
                    }
          }
6.  return  optimal_pair
```

Generation of an encrypted matrix

After a square matrix is obtain, a random matrix of the same dimensions is generated. This key matrix is very large, and has random values in the form of doubles, making it very difficult to guess (Nadir et al.)[7]. This key matrix is then multiplied with the generated square matrix, to obtain the encrypted matrix.

Pseudocode for Encryption

```
encrypt_wave(n,a)
Input:n, a                    //a is the .wav file
Output: Array enc
Variables: Integer flag,n
           Array  optimal_pair, a_reshaped, sq_wave, key, enc

1.  optimal_pair ← find_optimalpair(n)
2.  Interchange the values of optimal_pair[0] and optimal_pair[1]
3.   a_reshaped  ←  reshape(a,optimal_pair[1],optimal_pair[2])
4.  flag ← 0
5.  if optimal_pair[1] > optimal_pair[2]
    {
          sq_wave←horzcat(a_reshaped,zeros(optimal_pair[1],optimal_pair[1]-optimal_pair[2]))
                                                          //  Padding zeros to the matrix
    }
    else
    {
          flag ← 1
          a_reshaped ← transpose(a_reshaped)          //  Transposing the matrix
          sq_wave←horzcat(a_reshaped,zeros(optimal_pair[2],optimal_pair[2]-optimal_pair[1]))
                                                    //  Padding zeros to the matrix
    }

6.  key  ← rand(size(sq_wave))                        // Generating random key
7.  enc  ←  sq_wave*key                  `       // Encrypting the wave by multiplying to key
8.  return enc
```

The first ten seconds of the ECG .wav file, once encrypted, is as shown in Figure 4.

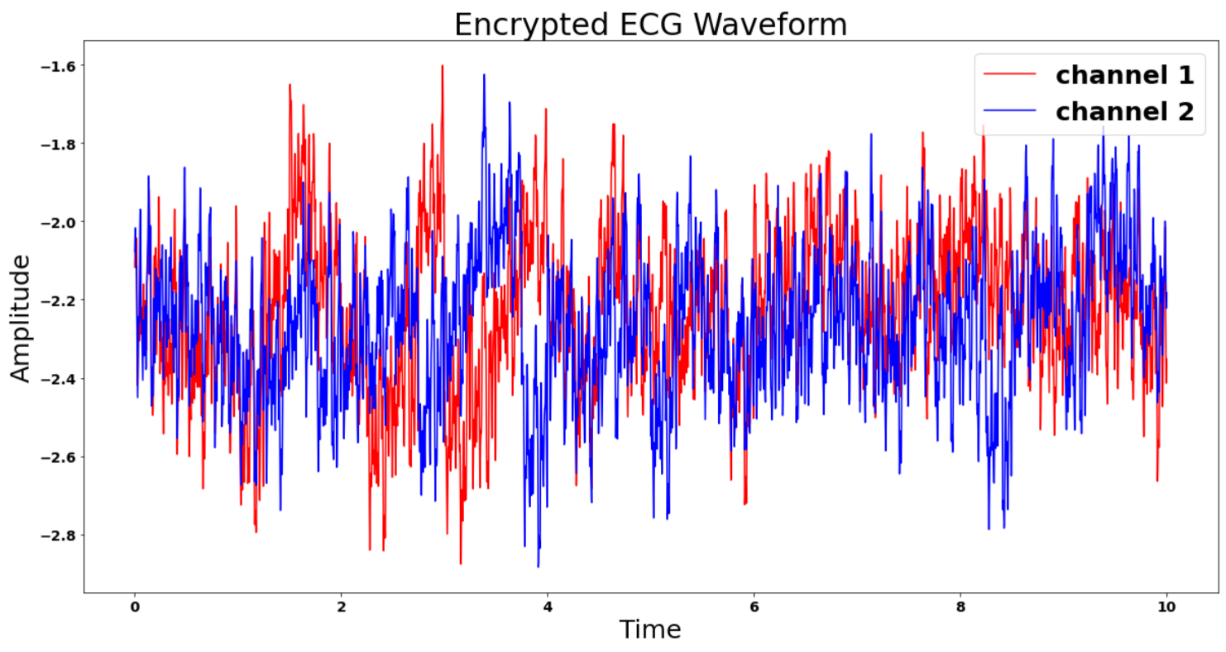

Figure 4: Ten seconds of the encrypted ECG Waveform

Pseudocode for Decryption

decrypt_wave(enc,key,flag)
Input*: enc,key,flag*
Output: *Array a*
Variables: *Integer flag*
Array enc, key,dec,a_stripped,a_reshaped

```
1.  dec ← enc/key                                  // Decrypting the wave using the key
2.  Strip the zeroes in dec and store in a_stripped
                                                   // Removing the padded zeros
3.  if flag=1
    {
            a_reshaped ←  transpose(a_stripped)
    }
4.  Reshape a_stripped such that it has only two columns and store it in a
                                              // Two channel wave contains only 2 columns
5.  return a
```

Efficiency of Proposed Methodology

The time taken for the encryption and decryption process is a key factor in the efficiency of a cryptographic technique. Ten-second samples from the ECG waveform are taken, and the average time taken for encryption and decryption is computed using the existing technique of padding multiple columns of zeroes to a single-column matrix, as well as the method of rearranging the columns such that a minimum number of zeroes is padded. The computation time in seconds for both methods is presented in **Table 1.**

Table 1: Comparison of Average Computation Time of different methodologies

Process	Average Computation Time (in seconds)	
	Method of padding zeroes to a column matrix	Method of optimizing matrix shape before padding zeroes
Encryption	7.4301	0.0000469
Decryption	35.7486	0.0001865

There is a clear decrease in the time taken for encryption and decryption, due to the overall decrease in the matrix's size.

Evaluation Metrics

In order to analyze the similarity between the original and decrypted waveforms, several metrics can be utilized. Histograms measure how robust the systems are against attacks. The better the system, the encrypted values should have a normal distribution (Algarni et al.)[1]. The Mean Square Error can also be calculated between the original and decrypted data to evaluate any possible error.

Histogram

In Figure 5, the distribution of the encrypted values is shown in the histogram. The plot shows a roughly normal distribution.

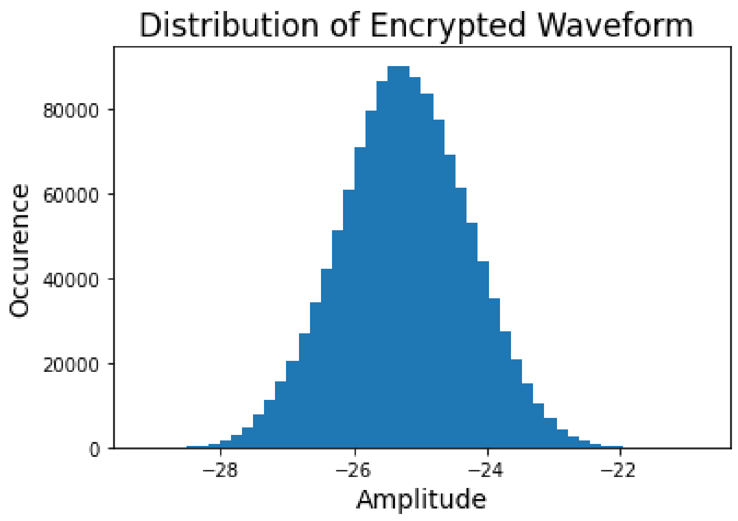

Figure 5: Histogram of distribution of encrypted waveform values

Mean Squared Error

The Mean Squared Error (MSE) gives the average squared difference between the actual and the observed values. The MSE is computed with the encrypted and decrypted values, to measure the extent of error produced during decryption. Figure 6 shows a plot of the Mean Squared Error of the ECG waveform. The Mean Squared Error(MSE) is given by Equation 1. It can be seen that the error between the original and the decrypted values is very minimal.

$$MSE = \frac{1}{n}\sum_{k=1}^{n}(x_i - \overline{x_i})^2 \qquad -(1)$$

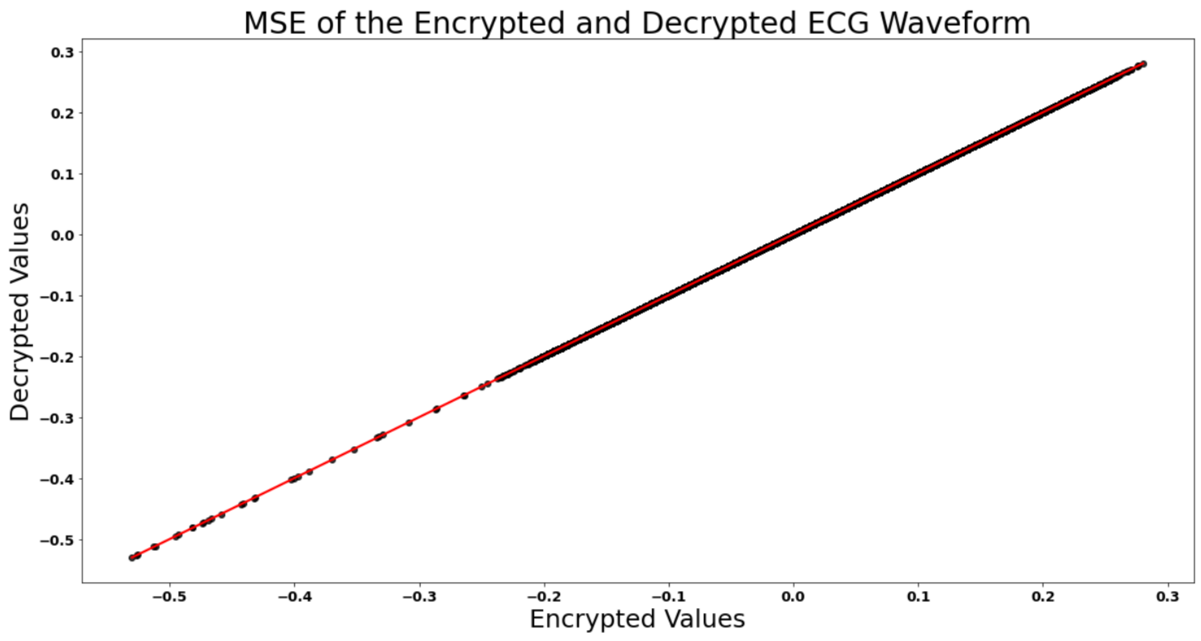

Figure 6: Plot for Mean Squared Error of encrypted and decrypted waveforms

Conclusion

In this paper, a method has been proposed to make encryption and decryption using matrices less time-consuming. It is also a solution with very high levels of accuracy. This method can also be extended for other encryption techniques that use matrices for encryption, where there is a necessity for square matrices. The padding of zeroes to make the matrix square is largely minimized. The random key matrix generated is also of a large order, even after minimization of the the zeroes padding, and the random values within it are doubles, making it secure, and thereby difficult to decipher. This method facilitates the secure transmission of ECG waveforms, without compromising on security. It is able to effectively encrypt and decrypt .wav files of longer durations, without loss of data.

References

1.Algarni, Abeer D., et al. "Encryption of ECG Signals for Telemedicine Applications." Multimedia Tools and Applications, vol. 80, 2021.

2.Al-laham, Mohamad M., et al. "A Method for Encrypting and Decrypting Wave Files." International Journal of Network Security and its Applications (IJNSA), vol. 10, no. 4, 2018.

3. Bai, Tong, et al. "A Lightweight method of Data Encryption in BANs using Electrocardiogram signal." Elsevier, Future Generation Computer Systems, 2019.

4. Cresswell, Kathrin, et al. The Future of Medical Informatics, Key Advances in Clinical Informatics. vol. Chapter 20, 2017.

5. Hameed, Mustafa Emad, et al. "Comparative Study of Several Modes of AES Algorithm for Encryption of ECG Biomedical Signal." International Journal of Electrical and Computer Engineering (IJECE), vol. 9, no. 6, pp. 4850-4859.

6. Moody, GB, and Mark RG. "The impact of the MIT-BIH Arrhythmia Database." IEEE Engineering in Medicine and Biology, vol. 20, no. 3, 2001, pp. 45-50.

7. Nadir, Jihad, et al. "A Technique to Encrypt-Decrypt Stereo Wave Files." International Journal of Computer and Information Technology, vol. 5, no. 5, 2016.

8. Ojha, Durgesh Kumar, and Monica Subashini. "Analysis of Electrocardiograph (ECG) Signal for the Detection of Abnormalities Using MATLAB." International Journal of Biomedical and Biological Engineering, vol. 8, no. 2, 2014.

9. Sattar, Yasar, and Lovely Chhabra. Electrocardiogram. Treasure Island (FL): StatPearls Publishing, 2022.

10. Upadhyay, Rahul H. "Study of Encryption and Decryption of Wave Files in Image Formats." arXiv preprint arXiv:1307.6711, 2013.