Lampiran 1. Proses FFT

```
// compute the FFT of x[]
        public static Complex[] fft(Complex[] x) {
            int N = x.Length;

            // base case
            if (N == 1) return new Complex[] { x[0] };

            // radix 2
            if (N % 2 != 0) { throw new ArithmeticException("N is not a power of
2"); }

            // fft of even terms
            Complex[] even = new Complex[N/2];
            for (int k = 0; k < N/2; k++) {
                even[k] = x[2*k];
            }
            Complex[] q = fft(even);

            // fft of odd terms
            Complex[] odd  = even;
            for (int k = 0; k < N/2; k++) {
                odd[k] = x[2*k + 1];
            }
            Complex[] r = fft(odd);

            // combine
            Complex[] y = new Complex[N];
            for (int k = 0; k < N/2; k++) {
                double kth = -2 * k * Math.PI / N;
                Complex wk = new Complex(Math.Cos(kth), Math.Sin(kth));
                y[k]       = q[k].plus(wk.times(r[k]));
                y[k + N/2] = q[k].minus(wk.times(r[k]));
            }

          return y;
        }
```

Lampiran 2. Proses IFFT

```
// compute the inverse FFT of x[]
        public static Complex[] ifft(Complex[] x) {
            int N = x.Length;
            Complex[] y = new Complex[N];

            // conjugate
            for (int i = 0; i < N; i++) {
                y[i] = x[i].conjugate();
            }

            // compute forward FFT
            y = fft(y);

            // conjugate again
            for (int i = 0; i < N; i++) {
                y[i] = y[i].conjugate();
            }

            // divide by N
            for (int i = 0; i < N; i++) {
               y[i] = y[i].conjugate();
                y[i] = y[i].times(1.0 / N);
            }

            return y;

        }
```

Lampiran 3. Proses *Random* Generator (LCG)

```
public double[] randGenerator(double a, double c, double m, int count)
        {

            double[] X_ = new double[count];
            X_[0] = 0.001;
            for (int i = 1; i < count; i++)
            {
                X_[i] = (a * X_[i - 1] + c) % m;
                X_[i] = X_[i] / m;
            }
            return X_;
        }
```

Lampiran 4. Proses Penyisipan Pesan

```
public void Embeding(String messageStream, String keyStream,
DevComponents.DotNetBar.Controls.ProgressBarX progress, long dataLength)
        {
            messageStream = Modulation(Marker + messageStream, keyStream, 1);
            BIN_Utils.ASCII phrase = new BIN_Utils.ASCII(messageStream);

            int[] messageAsBits = phrase.getBinaryBitArray();


            byte[] waveBuffer = new byte[bytesPerSample];

            int bytesRead = 0;
            int nbChannels = sourceStream.Header.getChannels();

            int totalBytes = (int)sourceStream.Header.nbSamples() * nbChannels;
                    double[] aout = new double[totalBytes];
            int bytesToRead = 4096 * 2; //  2^n
                    double[] audioData = new double[totalBytes];
            getAudioSamples(0, totalBytes, audioData);

            if (totalBytes / bytesToRead < messageAsBits.Length)
            {
                throw new MissingFieldException("The audio file is too short for
the message.. "  );
                    }
            int currentBit = 0;

            progress.Visible = true;
            progress.Maximum = messageAsBits.Length;
            progress.Value = 0;
            while (bytesRead < totalBytes && currentBit < messageAsBits.Length)
            {

                if (totalBytes - bytesRead < bytesToRead)
                {
                    bytesToRead = totalBytes - bytesRead;
                }
                double[] samples = new double[bytesToRead];

                for (int i = 0; i < samples.Length; i++)
                {
                    samples[i] = audioData[bytesRead + i] ;

                }
                bytesRead += bytesToRead;
                double[] channelData = new double[samples.Length / 2];
                getChannelSamples(0, samples, channelData);

                double[,] freqMag = FFT.FFT.getMag(channelData, (int)
sourceStream.Header.sampleRate);

                channelData = FFT.FFT.correctDataLength(channelData);
                FFT.Complex[] complexData = new FFT.Complex[channelData.Length];
                for (int i = 0; i < channelData.Length; i++)
                {
                    complexData[i] = new FFT.Complex(channelData[i], 0);
                }
```

```
                FFT.Complex[] complexMags = FFT.FFT.fft(complexData);

                double[] freqs = FFT.FFT.getFreqs(complexData.Length, (int)
sourceStream.Header.sampleRate);

                double tresholdAmp = 0;
                for (int i = 0; i < freqMag.GetLength(0); i++)
                {
                    if (   Math.Abs(freqMag[i,1]) > tresholdAmp)
                    {
                        tresholdAmp = freqMag[i,1];
                    }
                  //  System.Console.WriteLine(i + " " + freqMag[i, 1]);
                }
                Boolean isRest = false;
                if (tresholdAmp < .01)
                {
                    isRest = true;
                }

                if (messageAsBits[currentBit] == 1 && isRest == false)
                {
                    // for ifft
                    for (int i = 0; i < freqs.Length; i++)
                    {
                        if (Math.Abs(Math.Abs(freqs[i]) - 20000) < 5)
                        {
                            // changing  freq
                            complexMags[i] = new FFT.Complex(Alpha *
channelData.Length, 0);

                        }
                    }

                    // IFFT
                    FFT.Complex[] ifft = FFT.FFT.ifft(complexMags);

                    //change ifft data complex to real.
                    double[] ifftReal = new double[ifft.Length];

                    double[] spreadNoise = randGenerator(Alpha, Cr, totalBytes,
ifft.Length); ;
                    for (int i = 0; i < ifftReal.Length; i++)
                    {
                        ifftReal[i] = ifft[i].getReal();
                        ifftReal[i] += spreadNoise[i];
                       // System.Console.WriteLine(ifftReal[i]);
                    }

                    appendOutput(getAllSamples(ifftReal), bytesRead -
bytesToRead, aout);

                    currentBit++;
                }
                else if (messageAsBits[currentBit] == 0 && isRest == false)
                {
                    appendOutput(samples, bytesRead - bytesToRead, aout);
                    currentBit++;
                }
                else if (isRest == true)
```

```
                {
                    appendOutput(samples, bytesRead - bytesToRead, aout);
                }

                progress.Value = (int) currentBit;
            }

            progress.Visible = false;
                    if (bytesRead<totalBytes) {
                           double[] leftoverData = new double[totalBytes-
bytesRead];
                           for (int i = 0 ; i<leftoverData.Length ; i++) {
                                  leftoverData[i] = audioData[bytesRead+i];
                           }
                           appendOutput(leftoverData, bytesRead, aout);
                    }


            long numBytes = aout.Length * (
sourceStream.Header.getSampleSizeInBits() / 8);
            waveBuffer = new byte[numBytes];

            // Convert doubles to bytes
            sourceStream.encodeSamples(aout, waveBuffer, aout.Length);
            destinationStream.Write(waveBuffer, 0, waveBuffer.Length);

            System.Console.WriteLine("is done");
            destinationStream.SetLength(dataLength);

        }
```

Lampiran 5. Proses Ekstraksi Pesan

```
public String Extractor(String keyStream, int maxBit,
DevComponents.DotNetBar.Controls.ProgressBarX progress)
        {

            int[] messageAsBits = new int[maxBit];
            byte[] waveBuffer = new byte[bytesPerSample];

            int bytesRead = 0;
            int nbChannels = sourceStream.Header.getChannels();

            int totalBytes = (int)sourceStream.Header.nbSamples() * nbChannels;
            double[] aout = new double[totalBytes];
            int bytesToRead = 4096 * 2; //  2^n
            double[] audioData = new double[totalBytes];
            getAudioSamples(0, totalBytes, audioData);

            String[] messageAsBytes = new String[totalBytes / bytesToRead];
            int currentCharIndex = 0;
            int bitsSaved = 0;
            progress.Visible = true;
            progress.Maximum = messageAsBits.Length;
            progress.Value = 0;
            while (bytesRead < totalBytes)
            {
               // progress.Value = (int)currentCharIndex;

                if (totalBytes - bytesRead < bytesToRead)
                {
                    bytesToRead = totalBytes - bytesRead;
                }

                double[] samples = new double[bytesToRead];
                for (int i = 0; i < samples.Length; i++)
                {
                    samples[i] = audioData[bytesRead + i];
                }
                bytesRead += bytesToRead;
                double[] channelData = new double[samples.Length / 2];
                getChannelSamples(0, samples, channelData);

                channelData = FFT.FFT.correctDataLength(channelData);

                double[,] freqMag = FFT.FFT.getMag(channelData, (int)
sourceStream.Header.sampleRate);

                double tresholdAmp = 0;
                for (int i = 0; i < freqMag.GetLength(0); i++)
                {
                    if (Math.Abs(freqMag[i, 1]) > tresholdAmp)
                    {
                        tresholdAmp = freqMag[i, 1];
                    }
                }
                Boolean isRest = false;
                if (tresholdAmp < .01)
                {
                    isRest = true;
```

```
                }

                double ampToTest = 0;
                if (!isRest)
                {
                    for (int i = 0; i < freqMag.GetLength(0); i++)
                    {
                        if (Math.Abs(Math.Abs(freqMag[i, 0]) - 20000) < 5)
                        {
                            ampToTest = freqMag[i, 1];
                        }
                    }
                }

                if (!isRest)
                {

                    if (ampToTest > .009)
                    {
                        if (messageAsBytes[currentCharIndex] == null)
                        {
                            messageAsBytes[currentCharIndex] = "1";
                        }
                        else
                        {
                            messageAsBytes[currentCharIndex] = "1" +
messageAsBytes[currentCharIndex]; //adding 1
                        }

                    }
                    else
                    {
                        if (messageAsBytes[currentCharIndex] == null)
                        {
                            messageAsBytes[currentCharIndex] = "0";
                        }
                        else
                        {
                            messageAsBytes[currentCharIndex] = "0" +
messageAsBytes[currentCharIndex]; //adding  0
                        }
                    }
                    Console.WriteLine(messageAsBytes[currentCharIndex]);
                    bitsSaved++;
                    if (bitsSaved % 8 == 0)
                    {
                        if (messageAsBytes[currentCharIndex].Equals("00000000"))
                        {
                            System.Console.WriteLine("The message is over.");
                            break; //is done
                        }
                        currentCharIndex++;
                    }
                }

            }

            progress.Visible = false;
            String hiddenMessage = "";
            hiddenMessage = constructMessage(messageAsBytes);
```

```
            System.Console.WriteLine("Msg " + hiddenMessage);
            hiddenMessage = hiddenMessage.Remove(hiddenMessage.Length - 1);
            hiddenMessage = Modulation(hiddenMessage, keyStream.ToString(), 0);
            if (hiddenMessage.Substring(0, Marker.Length) == Marker)
            {
                hiddenMessage = hiddenMessage.Substring(Marker.Length);
            }
            else
            {
                hiddenMessage = "-";
            }

            return hiddenMessage;
        }

    }
}
```