```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Timers;
using System.Text;
using System.Threading.Tasks;

namespace PlayfairCipher
{
    class Program
    {
        protected static int origRow;
        protected static int origCol;

        protected static void WriteAt(string s, int x, int y)
        {
            try
            {
                Console.SetCursorPosition(origCol + x, origRow + y);
                Console.Write(s);
            }
            catch (ArgumentOutOfRangeException e)
            {
                Console.Clear();
                Console.WriteLine(e.Message);
            }
        }

        static void Main(string[] args)
        {
            #region Encription
            Console.Write("Podaj tekst jawny: ");
            string publicText = Console.ReadLine().ToUpper().Replace(" ", "");
            char[] publicTextCharTab = new char[publicText.Length];
            publicTextCharTab = publicText.ToArray();
            Console.Write("Podaj klucz szyfrujący: ");
            string key = Console.ReadLine().ToUpper().Replace(" ", "");
            char[] keyCharTab = new char[key.Length];
            keyCharTab = key.ToList().Distinct().ToArray();
            PlayfairTab(keyCharTab);
            publicTextCharTab = AddictionalLetter(publicTextCharTab);
            int[] typeOfCodingTab = new int[publicTextCharTab.Length / 2];
            typeOfCodingTab = TypeOfCoding(Coordinates(PlayfairTab(keyCharTab), publicTextCharTab));
            int[] encriptedCoordinate = new int[Coordinates(PlayfairTab(keyCharTab), publicTextCharTab).Length];
            encriptedCoordinate = PlayfairCypherEncription(PlayfairTab(keyCharTab), Coordinates(PlayfairTab(keyCharTab), publicTextCharTab), typeOfCodingTab);
            string encriptedString = new string(EncriptedCoordinatesToCharTab(encriptedCoordinate,PlayfairTab(keyCharTab)));
            char[] EncriptedCharTab = new char[EncriptedCoordinatesToCharTab(encriptedCoordinate, PlayfairTab(keyCharTab)).Length];
            EncriptedCharTab = EncriptedCoordinatesToCharTab(encriptedCoordinate, PlayfairTab(keyCharTab));
            Console.WriteLine("\nKODOWANIE: ");
            Random rnd = new Random();

            Timer aTimer = new System.Timers.Timer();
            aTimer.Interval = 50;

            // Hook up the Elapsedd event for the timer.
            DateTime t = DateTime.Now;
            aTimer.Elapsed += (sender, e) => OnTimedEvent(sender, e, EncriptedCharTab, t);

            // Have the timer fire repeated events (true is the default)
            aTimer.AutoReset = true;

            // Start the timer
            aTimer.Enabled = true;

            Console.ReadLine();

            #endregion

            #region Decription

            bool isCorrect = false;
            do
            {
                Console.WriteLine("Wybierz \n 1) Aby odszyfrować pierwotne hasło \n 2) Aby odszyfrować dowolne hasło");
                int choice = Int32.Parse(Console.ReadLine());
                if (choice==1)
                {
                    isCorrect = true;
                    string decriptedString = new string(DecriptedCoordinatesToCharTab(PlayfairCypherDecription(PlayfairTab(keyCharTab), CoordinatesDecription(PlayfairTab(keyCharTab), EncriptedCharTab), TypeOfCodingDecripted(CoordinatesDecription(PlayfairTab(keyCharTab), EncriptedCharTab))), PlayfairTab(keyCharTab)));
                    Console.WriteLine(decriptedString);
                }
                else
                {
                    if (choice==2)
                    {
                        Console.Write("Podaj hasło do odszyfrowania: ");
                        string encriptedStringDecoding = Console.ReadLine();
                        char[] encriptedTextDecoding = new char[encriptedStringDecoding.Length];
                        isCorrect = true;
                        string decriptedString = new string(DecriptedCoordinatesToCharTab(PlayfairCypherDecription(PlayfairTab(keyCharTab), CoordinatesDecription(PlayfairTab(keyCharTab), EncriptedCharTab), TypeOfCodingDecripted(CoordinatesDecription(PlayfairTab(keyCharTab), EncriptedCharTab))), PlayfairTab(keyCharTab)));
                        Console.WriteLine(decriptedString);
                    }
                    else
                    {
                        isCorrect = false;
                        Console.WriteLine("Nie poprawnie wybrana opcja.");
                    }
                }
            } while (!isCorrect);

            #endregion
        }

        #region EncriptionFunction

        private static void OnTimedEvent(Object source,
          System.Timers.ElapsedEventArgs e, char[] encriptedCharTab, DateTime
          t)
        {
            TimeSpan dtts = new TimeSpan(t.Day, t.Hour, t.Minute,t.Second);
            int dttsTempSec = (int)dtts.TotalSeconds;
            TimeSpan ts = new TimeSpan(e.SignalTime.Day, e.SignalTime.Hour,
              e.SignalTime.Minute, e.SignalTime.Second);
            int tempSec = (int)ts.TotalSeconds;
            t =t.Add(ts);

            Random rnd = new Random();
            for (int i = tempSec - dttsTempSec; i < encriptedCharTab.Length; i
              ++)
            {
                WriteAt(((char)rnd.Next(65,90)).ToString(), i, 4);
            }
            if (tempSec - dttsTempSec<encriptedCharTab.Length)
            {
                WriteAt(encriptedCharTab[tempSec - dttsTempSec].ToString(),
                  tempSec - dttsTempSec, 4);
            }

        }

        public static char[,] PlayfairTab(char[] keyCharTab)
        {
            char[] AlphabetTab = new char[26];
            AlphabetTab[0] = 'A';
            for (int i = 1; i < AlphabetTab.Length; i++)
            {
                AlphabetTab[i] = (char)((int)AlphabetTab[0] + i);
            }
            char[] PlayfairTab1D = new char[keyCharTab.Length +
              AlphabetTab.Length];
            PlayfairTab1D = keyCharTab.Concat(AlphabetTab).ToArray().ToList
              ().Distinct().ToArray().Where(x => x != 'J').ToArray();
            char[,] PlayfairTab2D = new char[5, 5];
            for (int i = 0; i < 5; i++)
            {
                for (int j = 0; j < 5; j++)
                {
                    PlayfairTab2D[i, j] = PlayfairTab1D[5 * i + j];
                }
            }
            return PlayfairTab2D;
        }

        public static char[] AddictionalLetter(char[] publicTextCharTab)
        {
            int index = -2;
            do
            {
                index = ReturnDoubleLetterIndex(publicTextCharTab);
                if (index != -1)
                {
                    List<char> tempCharList = publicTextCharTab.ToList();
                    tempCharList.Insert(index, 'X');
                    publicTextCharTab = tempCharList.ToArray();
                }
            } while (index != -1);
            if (publicTextCharTab.Length % 2 == 1)
            {
                List<char> tempCharList = publicTextCharTab.ToList();
                tempCharList.Insert(publicTextCharTab.Length, 'X');
                publicTextCharTab = tempCharList.ToArray();
            }
            return publicTextCharTab;
        }

        public static int ReturnDoubleLetterIndex(char[] publicTextCharTab)
        {
            for (int i = 0; i < publicTextCharTab.Length; i += 2)
            {
                if (i + 1 < publicTextCharTab.Length)
                {
                    if (publicTextCharTab[i] == publicTextCharTab[i + 1])
                    {
                        return i + 1;
                    }
                }
            }
            return -1;
        }

        public static int[] Coordinates(char[,] playfairTab, char[]
          publicTextCharTab)
        {
            int[] coordinates = new int[(publicTextCharTab.Length) * 2];
            for (int k = 0; k < publicTextCharTab.Length; k++)
            {
                for (int i = 0; i < 5; i++)
                {
                    for (int j = 0; j < 5; j++)
                    {
                        if (publicTextCharTab[k] == playfairTab[i, j])
                        {
                            coordinates[2 * k] = i;
                            coordinates[2 * k + 1] = j;
                        }
                    }
                }
            }
            return coordinates;
        }

        public static int[] TypeOfCoding(int[] coordinates)
        {
            int[] TypeOfCodingTab = new int[coordinates.Length / 4];
            for (int i = 0; i < coordinates.Length / 4; i++)
            {
                if (4 * i + 3 < coordinates.Length)
                {
                    if (coordinates[4 * i] == coordinates[4 * i + 2])
                    {
                        TypeOfCodingTab[i] = 2;
                    }
                    else
                    {
                        if (coordinates[4 * i + 1] == coordinates[4 * i + 3])
                        {
                            TypeOfCodingTab[i] = 1;
                        }
                        else
                        {
                            TypeOfCodingTab[i] = 3;
                        }
                    }
                }
            }
            return TypeOfCodingTab;
        }

        public static int[] PlayfairCypherEncription(char[,] playfairTab, int[] coordinates, int[] typeOfCodingTab)
        {
            for (int i = 0; i < typeOfCodingTab.Length; i++)
            {
                if (typeOfCodingTab[i] == 1)
                {
                    coordinates[4 * i] = (coordinates[4 * i] + 1) % 5;
                    coordinates[4 * i + 2] = (coordinates[4 * i + 2] + 1) % 5;
                }
                else
                {
                    if (typeOfCodingTab[i] == 2)
                    {
                        coordinates[4 * i + 1] = (coordinates[4 * i + 1] + 1) % 5;
                        coordinates[4 * i + 3] = (coordinates[4 * i + 3] + 1) % 5;
                    }
                    else
```

```csharp
                    {
                        int temp = 0;
                        temp = coordinates[4 * i];
                        coordinates[4 * i] = coordinates[4 * i + 2];
                        coordinates[4 * i + 2] = temp;
                    }
                }
            }
            return coordinates;
        }

        public static char[] EncriptedCoordinatesToCharTab(int[] encriptedCoordinates, char[,] playfairCharTab)
        {
            char[] encriptedCharTab = new char[encriptedCoordinates.Length / 2];
            for (int i = 0; i < encriptedCoordinates.Length/2; i++)
            {
                encriptedCharTab[i] = playfairCharTab[encriptedCoordinates[2*i],encriptedCoordinates[2*i + 1]];
            }
            return encriptedCharTab;
        }

        #endregion

        #region DecriptionFunction

        public static int[] CoordinatesDecription(char[,] playfairTab, char[] encriptedCharTab)
        {
            int[] coordinatesDecripted = new int[(encriptedCharTab.Length) * 2];
            for (int k = 0; k < encriptedCharTab.Length; k++)
            {
                for (int i = 0; i < 5; i++)
                {
                    for (int j = 0; j < 5; j++)
                    {
                        if (encriptedCharTab[k] == playfairTab[i, j])
                        {
                            coordinatesDecripted[2 * k] = i;
                            coordinatesDecripted[2 * k + 1] = j;
                        }
                    }
                }
            }
            return coordinatesDecripted;
        }

        public static int[] TypeOfCodingDecripted(int[] coordinatesDecripted)
        {
            int[] TypeOfCodingTabDecripted = new int[coordinatesDecripted.Length / 4];
            for (int i = 0; i < coordinatesDecripted.Length / 4; i++)
            {
                if (4 * i + 3 < coordinatesDecripted.Length)
                {
                    if (coordinatesDecripted[4 * i] == coordinatesDecripted[4 * i + 2])
                    {
                        TypeOfCodingTabDecripted[i] = 2;
                    }
                    else
                    {
                        if (coordinatesDecripted[4 * i + 1] == coordinatesDecripted[4 * i + 3])
                        {
                            TypeOfCodingTabDecripted[i] = 1;
                        }
                        else
                        {
                            TypeOfCodingTabDecripted[i] = 3;
                        }
                    }
                }
            }
            return TypeOfCodingTabDecripted;
        }

        public static int[] PlayfairCypherDecription(char[,] playfairTab, int[] coordinatesDecripted, int[] typeOfCodingTabDecripted)
        {
            for (int i = 0; i < typeOfCodingTabDecripted.Length; i++)
            {
                if (typeOfCodingTabDecripted[i] == 1)
                {
                    if ((coordinatesDecripted[4 * i] - 1) % 5 < 0)
                    {
                        coordinatesDecripted[4 * i] = ((coordinatesDecripted[4 * i] - 1) % 5)+5;
                    }
                    else
                    {
                        coordinatesDecripted[4 * i] = ((coordinatesDecripted[4 * i] - 1) % 5);
                    }
                    if ((coordinatesDecripted[4 * i + 2] - 1) % 5 < 0)
                    {
                        coordinatesDecripted[4 * i + 2] = ((coordinatesDecripted[4 * i + 2] - 1) % 5)+5;
                    }
                    else
                    {
                        coordinatesDecripted[4 * i + 2] = (coordinatesDecripted[4 * i + 2] - 1) % 5;
                    }

                }
                else
                {
                    if (typeOfCodingTabDecripted[i] == 2)
                    {
                        if ((coordinatesDecripted[4 * i + 1] - 1) % 5 < 0)
                        {
                            coordinatesDecripted[4 * i + 1] = ((coordinatesDecripted[4 * i + 1 ] - 1) % 5) + 5;
                        }
                        else
                        {
                            coordinatesDecripted[4 * i + 1] = ((coordinatesDecripted[4 * i + 1] - 1) % 5);
                        }
                        if ((coordinatesDecripted[4 * i + 3] - 1) % 5 < 0)
                        {
                            coordinatesDecripted[4 * i + 3] = ((coordinatesDecripted[4 * i + 3] - 1) % 5) + 5;
                        }
                        else
                        {
                            coordinatesDecripted[4 * i + 3] = (coordinatesDecripted[4 * i + 3] - 1) % 5;
                        }
                    }
                    else
                    {
                        int temp = 0;
                        temp = coordinatesDecripted[4 * i];
                        coordinatesDecripted[4 * i] = coordinatesDecripted[4 * i + 2];
                        coordinatesDecripted[4 * i + 2] = temp;
                    }
                }
            }
            return coordinatesDecripted;
        }

        public static char[] DecriptedCoordinatesToCharTab(int[] coordinatesDecripted, char[,] playfairCharTab)
        {
            char[] decriptedCharTab = new char[coordinatesDecripted.Length / 2];
            for (int i = 0; i < coordinatesDecripted.Length / 2; i++)
            {
                decriptedCharTab[i] = playfairCharTab[coordinatesDecripted[2 * i], coordinatesDecripted[2 * i + 1]];
            }
            return decriptedCharTab;
        }

        #endregion
    }
}
```