Caesar Cipher Program Encryption and Decryption

Prepared By :

Mahmoud Youssef Mahmoud

20043261

Supervised By :

Prof. Shawkat K. Guirguis

Professor of Computer Science & Informatics

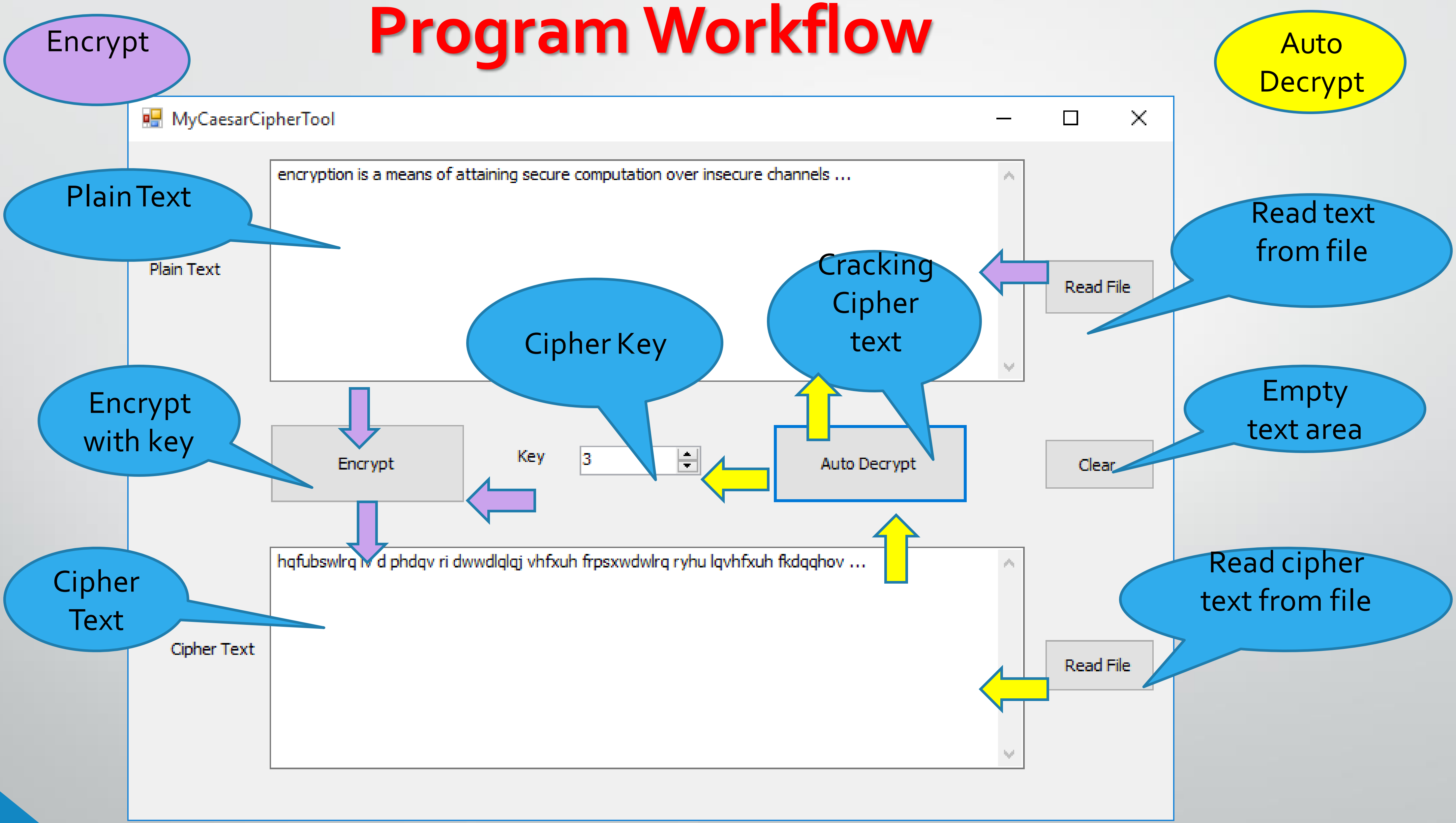
Information Technology Department

Institute of Graduate Studies & Research

Alexandria University

Contents

Program Workflow



encryption is a means of attaining secure computation over insecure channels ...

3

hqfubswlrq lv d phdqv ri dwwdlqlqj vhfxuh frpsxwdwlrq ryhu lqvhfxuh fkdqqhov ...

Program Workflow

Encryption Workflow:

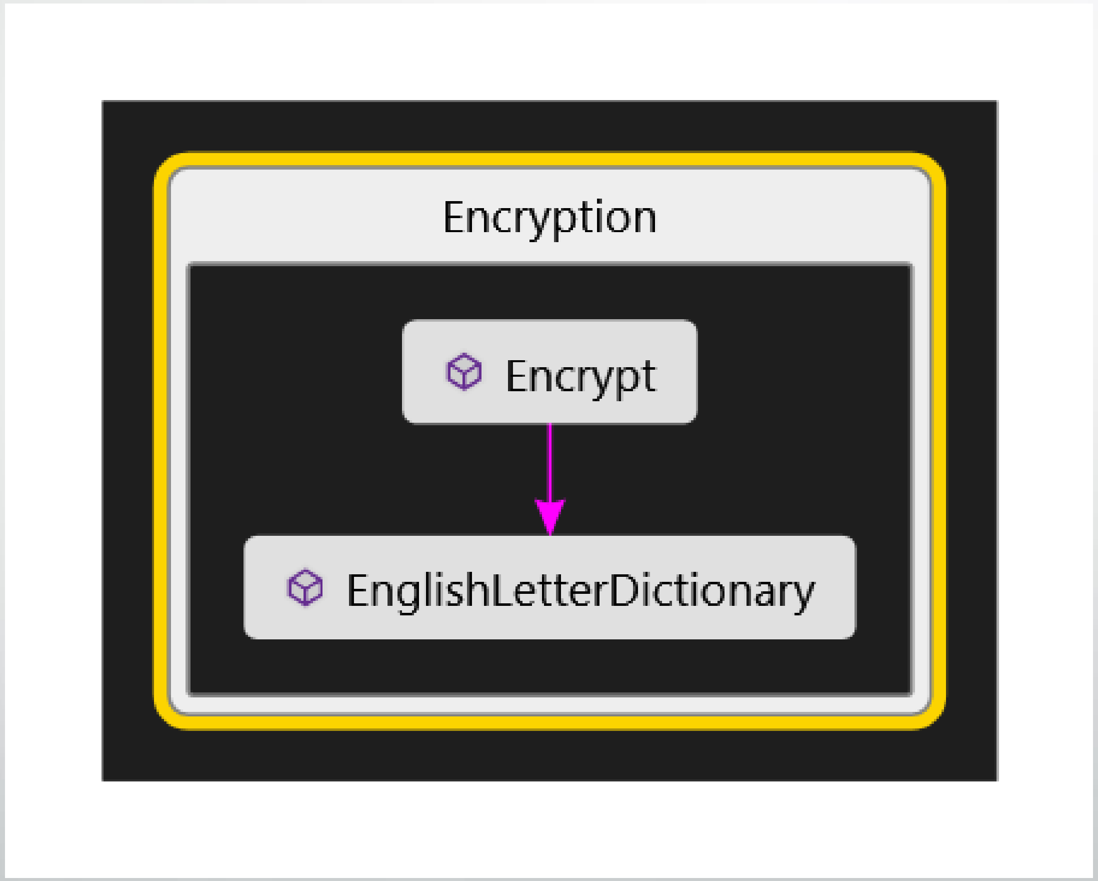
- Click Read File beside plain text area to read text file or type plain text in text box
- Enter cipher Key
- Press Encrypt button

Program Workflow

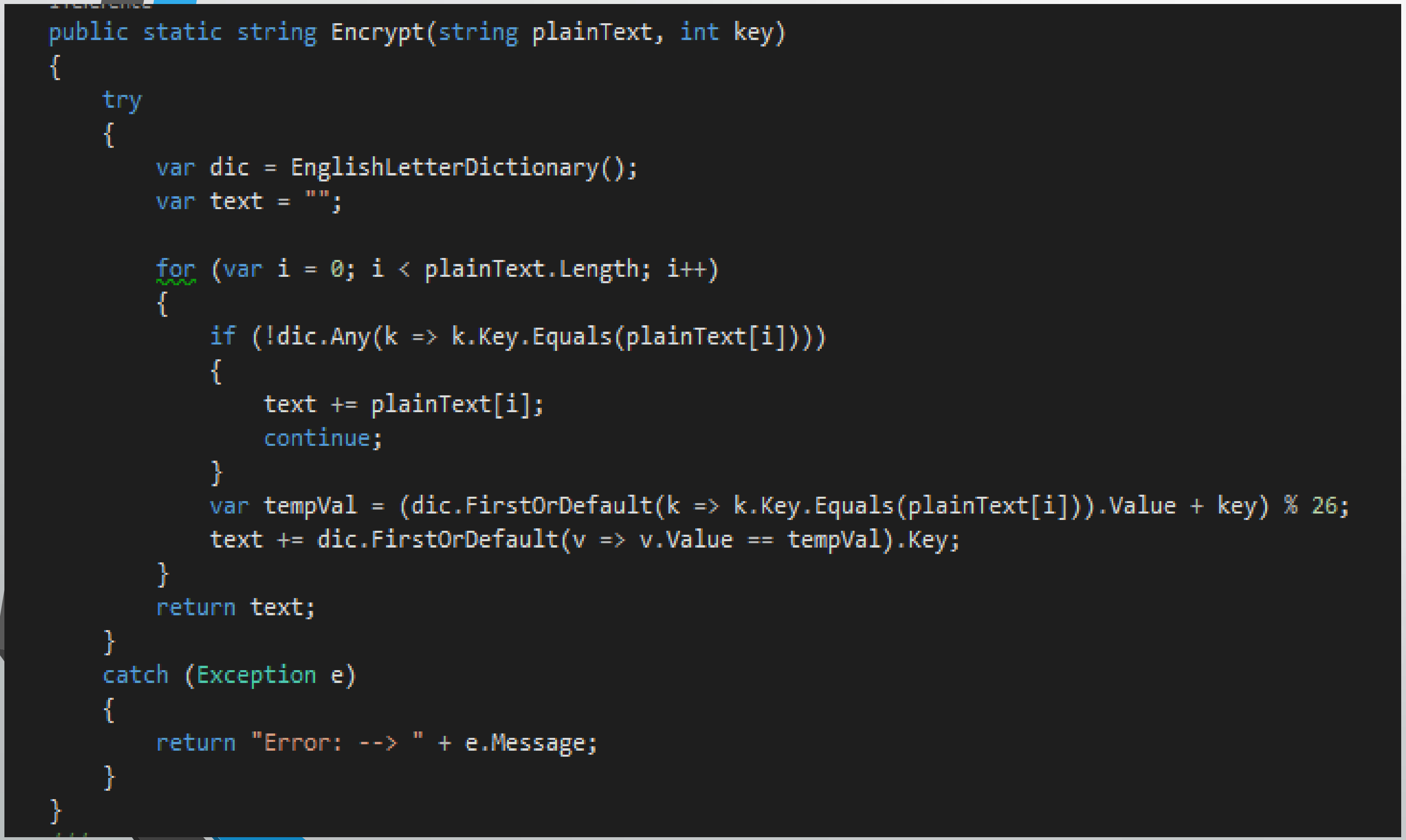
Decryption Workflow:

- Click Read File beside cipher text area to read text file or type plain text in text box
- Press Auto Decrypt button
- The program automatically detect key and write it in key number box
- The program automatically decrypt message and write Plain message

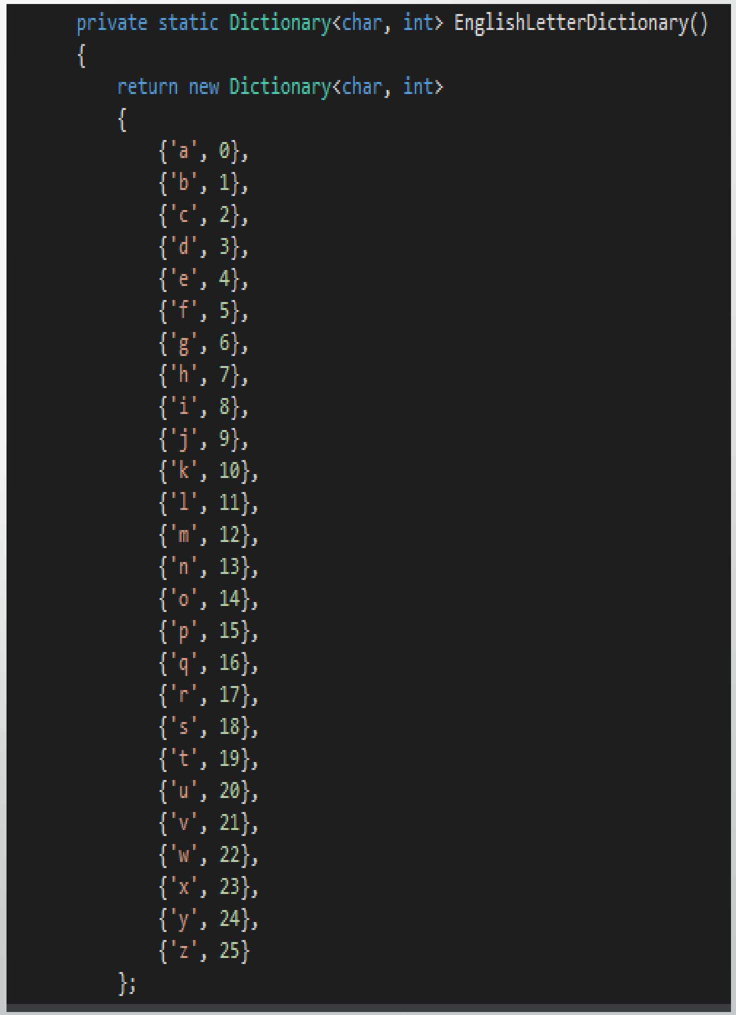
Encryption Function



Encryption Function



```
public static string Encrypt(string plainText, int key)
{
    try
    {
        var dic = EnglishLetterDictionary();
        var text = "";

        for (var i = 0; i < plainText.Length; i++)
        {
            if (!dic.Any(k => k.Key.Equals(plainText[i])))
            {
                text += plainText[i];
                continue;
            }
            var tempVal = (dic.FirstOrDefault(k => k.Key.Equals(plainText[i])).Value + key) % 26;
            text += dic.FirstOrDefault(v => v.Value == tempVal).Key;
        }
        return text;
    }
    catch (Exception e)
    {
        return "Error: --> " + e.Message;
    }
}
```

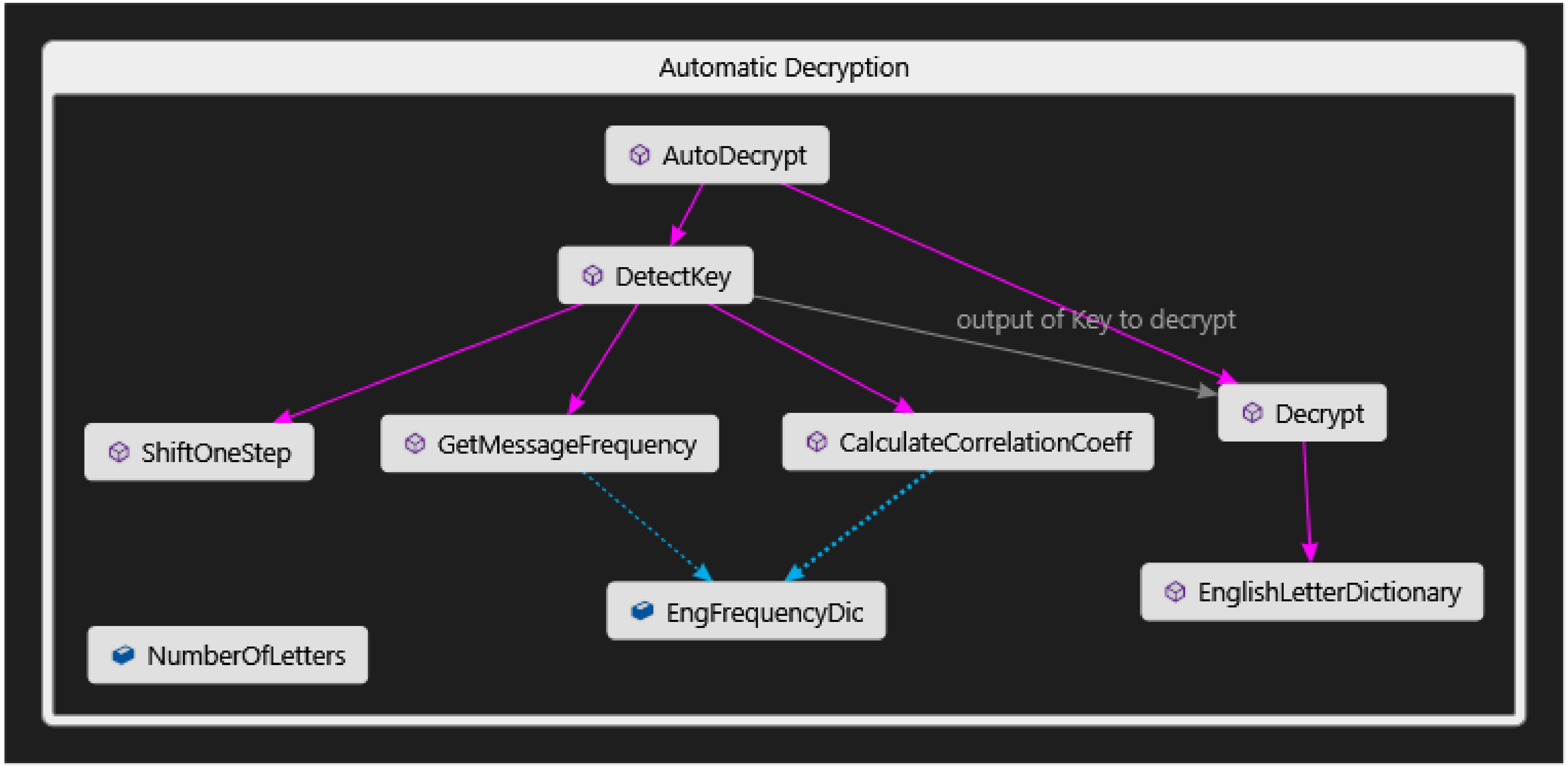


```
private static Dictionary<char, int> EnglishLetterDictionary()
{
    return new Dictionary<char, int>
    {
        {'a', 0},
        {'b', 1},
        {'c', 2},
        {'d', 3},
        {'e', 4},
        {'f', 5},
        {'g', 6},
        {'h', 7},
        {'i', 8},
        {'j', 9},
        {'k', 10},
        {'l', 11},
        {'m', 12},
        {'n', 13},
        {'o', 14},
        {'p', 15},
        {'q', 16},
        {'r', 17},
        {'s', 18},
        {'t', 19},
        {'u', 20},
        {'v', 21},
        {'w', 22},
        {'x', 23},
        {'y', 24},
        {'z', 25}
    };
```

Encryption Function

- This function take plain text and encryption key
- Initialize English Dictionary that contains all English letters and its equivalent numbers
- Search for each letter of the message in English Dictionary and get its equivalent number
- Sum letter number with given Key then take its modulus to 26
- Return equivalent latter for this number
- If message letter not found in dictionary its return same letter such as any special letters (?,!,1,2,…)
- Return Cipher message

Decryption Function



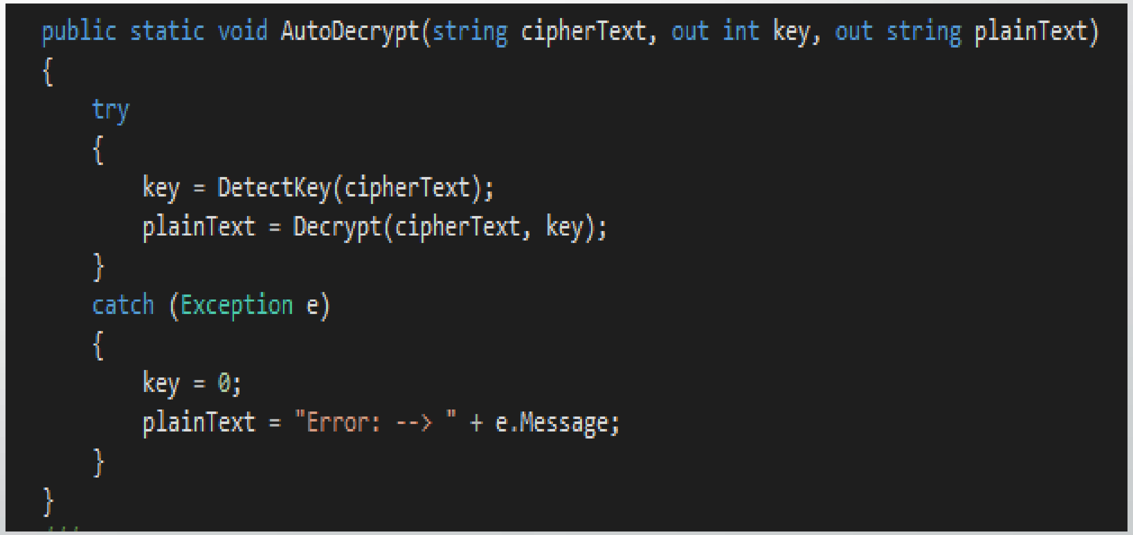
Decryption Function

- Cracking cipher text by Caesar Cipher Algorithm using Correlation Coefficient Method by:
- calculation message letter frequency distribution
- calculate Correlation between message letter frequency
- List and English letter frequency distribution
- shift message frequency list by one step and calculate Correlation again
- repeat this steps for 26 time number of English letters
- find max Correlation Coefficient
- get index of max Correlation Coefficient this is the KEY

use the KEY to Decrypt cipher message

Decryption Function

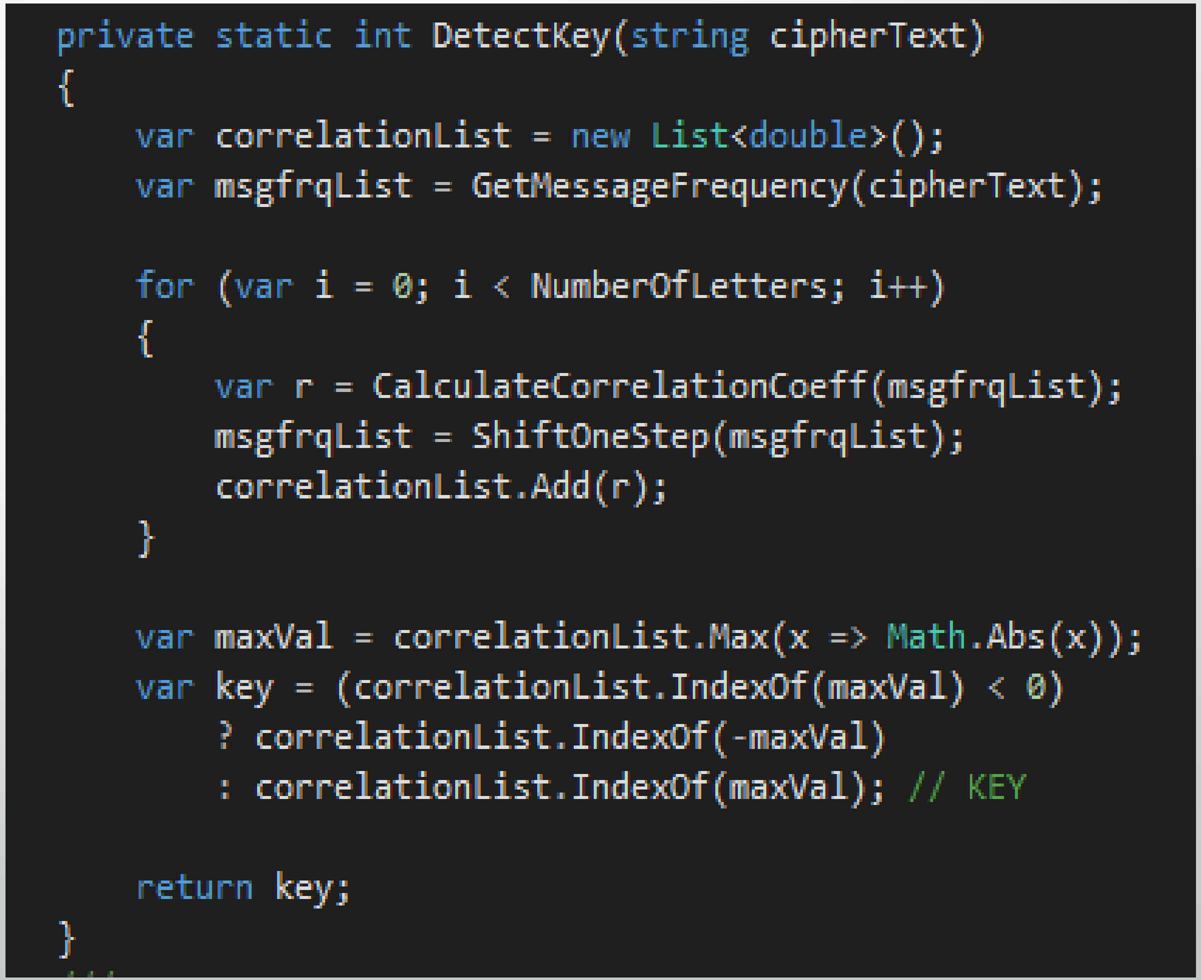
- To Crack cipher text first you have to detect Key
- DetectKey function use sub functions
- GetMessageFrequency()
- ShiftOneStep()
- CalculateCorrelationCoeff()
- Then use this key to decipher() message



```
public static void AutoDecrypt(string cipherText, out int key, out string plainText)
{
    try
    {
        key = DetectKey(cipherText);
        plainText = Decrypt(cipherText, key);
    }
    catch (Exception e)
    {
        key = 0;
        plainText = "Error: --> " + e.Message;
    }
}
```

Decryption Function

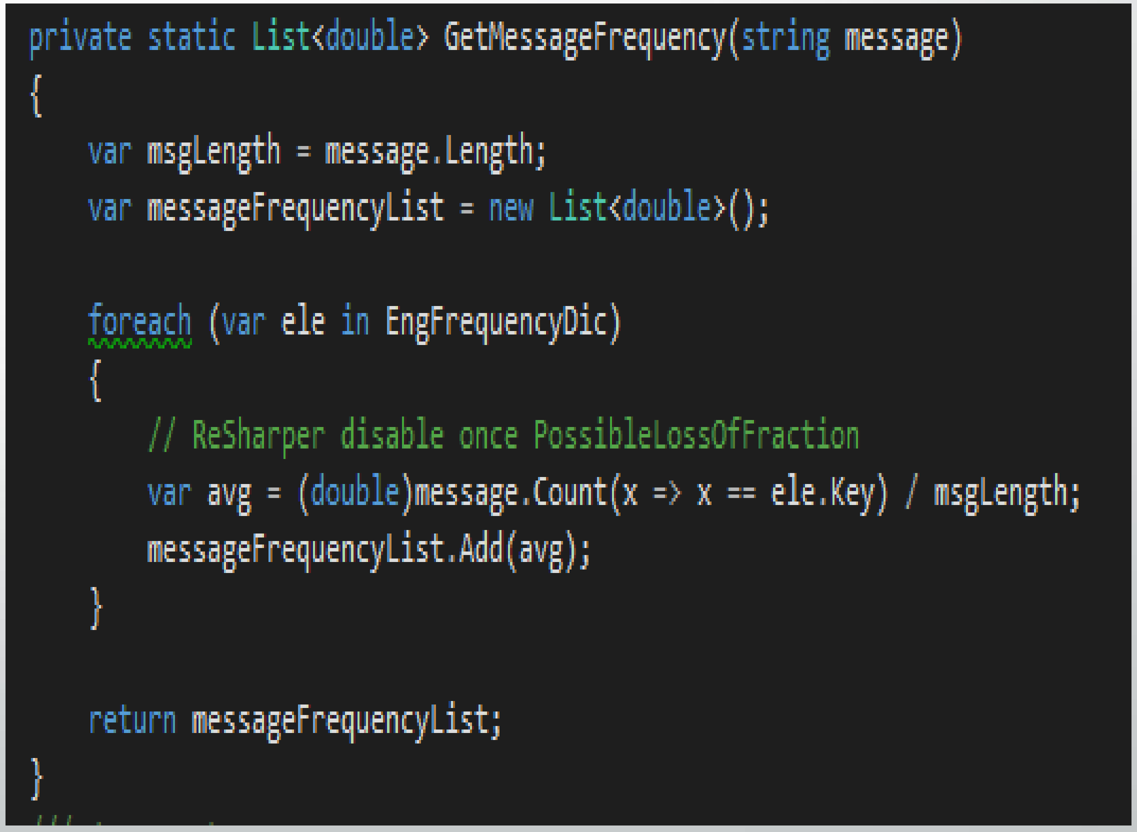
- Detect cipher key by calculation message letter frequency distribution
- calculate Correlation between message letter frequency List and English letter frequency distribution
- shift message frequency list by one step and calculate Correlation again
- repeat it for 26 time number of English letters
- find max Correlation Coefficient
- get index of max Correlation Coefficient this is the KEY



```
private static int DetectKey(string cipherText)
{
    var correlationList = new List<double>();
    var msgfrqList = GetMessageFrequency(cipherText);

    for (var i = 0; i < NumberOfLetters; i++)
    {
        var r = CalculateCorrelationCoeff(msgfrqList);
        msgfrqList = ShiftOneStep(msgfrqList);
        correlationList.Add(r);
    }

    var maxVal = correlationList.Max(x => Math.Abs(x));
    var key = (correlationList.IndexOf(maxVal) < 0)
        ? correlationList.IndexOf(-maxVal)
        : correlationList.IndexOf(maxVal); // KEY

    return key;
}
```

Decryption Function

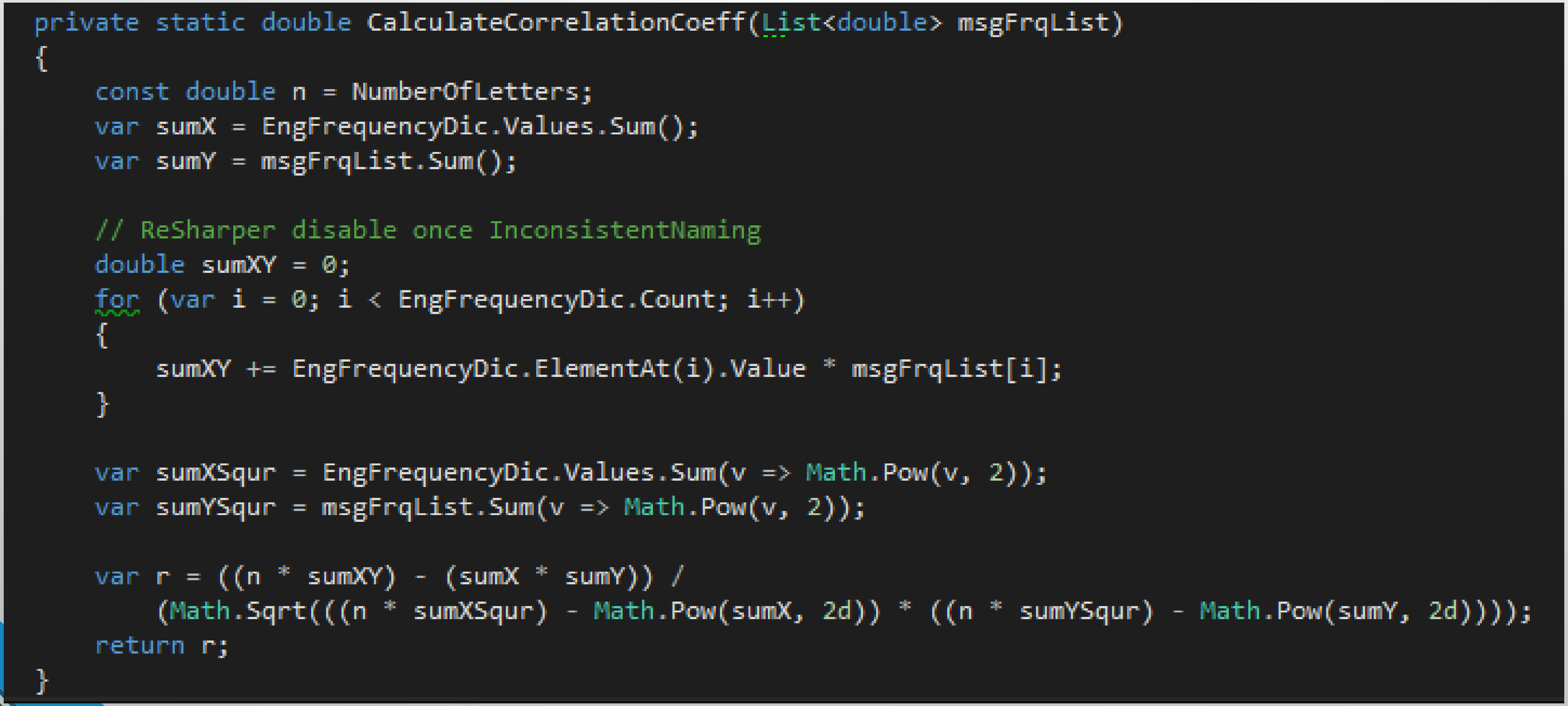
- GetMessageFrequency
- Calculate message letters frequency distribution
- By counting each letter in message and divide it with message length



```
private static List<double> GetMessageFrequency(string message)
{
    var msgLength = message.Length;
    var messageFrequencyList = new List<double>();

    foreach (var ele in EngFrequencyDic)
    {
        // ReSharper disable once PossibleLossOfFraction
        var avg = (double)message.Count(x => x == ele.Key) / msgLength;
        messageFrequencyList.Add(avg);
    }

    return messageFrequencyList;
}
```

Decryption Function

- Calculate Correlation Coefficient function

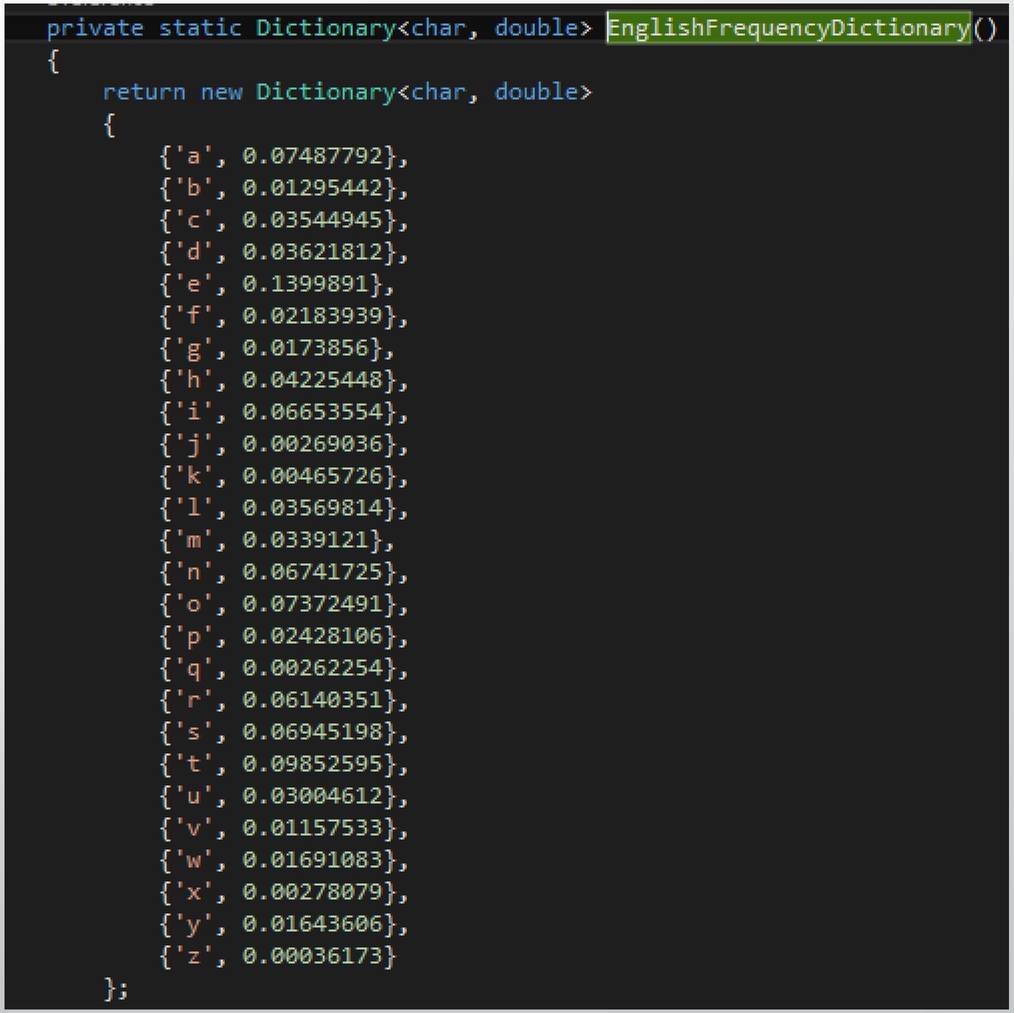


```
private static double CalculateCorrelationCoeff(List<double> msgFrqList)
{
    const double n = NumberOfLetters;
    var sumX = EngFrequencyDic.Values.Sum();
    var sumY = msgFrqList.Sum();

    // ReSharper disable once InconsistentNaming
    double sumXY = 0;
    for (var i = 0; i < EngFrequencyDic.Count; i++)
    {
        sumXY += EngFrequencyDic.ElementAt(i).Value * msgFrqList[i];
    }

    var sumXSqur = EngFrequencyDic.Values.Sum(v => Math.Pow(v, 2));
    var sumYSqur = msgFrqList.Sum(v => Math.Pow(v, 2));

    var r = ((n * sumXY) - (sumX * sumY)) /
        (Math.Sqrt(((n * sumXSqur) - Math.Pow(sumX, 2d)) * ((n * sumYSqur) - Math.Pow(sumY, 2d))));
    return r;
}
```

Decryption Function

- Calculate Correlation Coefficient function
- Calculate Correlation Coefficient function between each character frequency in message and its equivalent English letter frequency
- According to this Equation

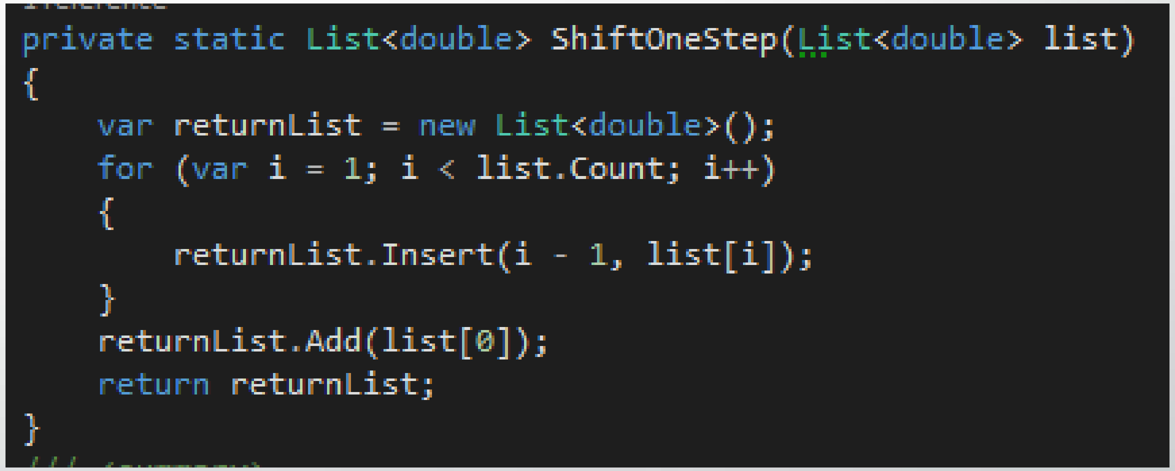
$$r = \frac{n\sum xy - \left(\sum x\right)\left(\sum y\right)}{\sqrt{n\left(\sum x^2\right) - \left(\sum x\right)^2}\sqrt{n\left(\sum y^2\right) - \left(\sum y\right)^2}}$$



```
private static Dictionary<char, double> EnglishFrequencyDictionary()
{
    return new Dictionary<char, double>
    {
        {'a', 0.07487792},
        {'b', 0.01295442},
        {'c', 0.03544945},
        {'d', 0.03621812},
        {'e', 0.1399891},
        {'f', 0.02183939},
        {'g', 0.0173856},
        {'h', 0.04225448},
        {'i', 0.06653554},
        {'j', 0.00269036},
        {'k', 0.00465726},
        {'l', 0.03569814},
        {'m', 0.0339121},
        {'n', 0.06741725},
        {'o', 0.07372491},
        {'p', 0.02428106},
        {'q', 0.00262254},
        {'r', 0.06140351},
        {'s', 0.06945198},
        {'t', 0.09852595},
        {'u', 0.03004612},
        {'v', 0.01157533},
        {'w', 0.01691083},
        {'x', 0.00278079},
        {'y', 0.01643606},
        {'z', 0.00036173}
    };
```

Decryption Function

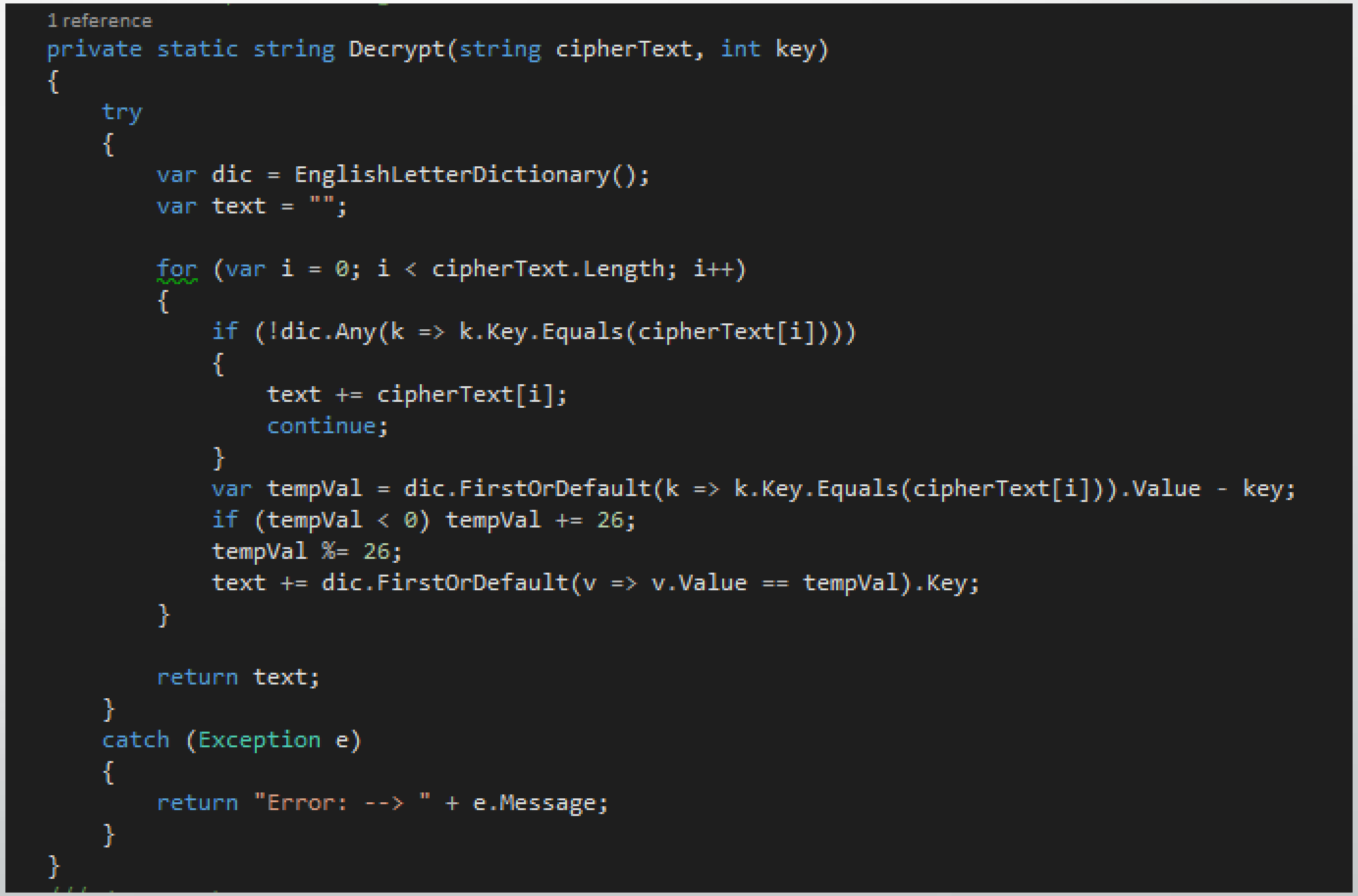
- ShiftOneStep
- shift list of message frequencies by one step



```
private static List<double> ShiftOneStep(List<double> list)
{
    var returnList = new List<double>();
    for (var i = 1; i < list.Count; i++)
    {
        returnList.Insert(i - 1, list[i]);
    }
    returnList.Add(list[0]);
    return returnList;
}
```

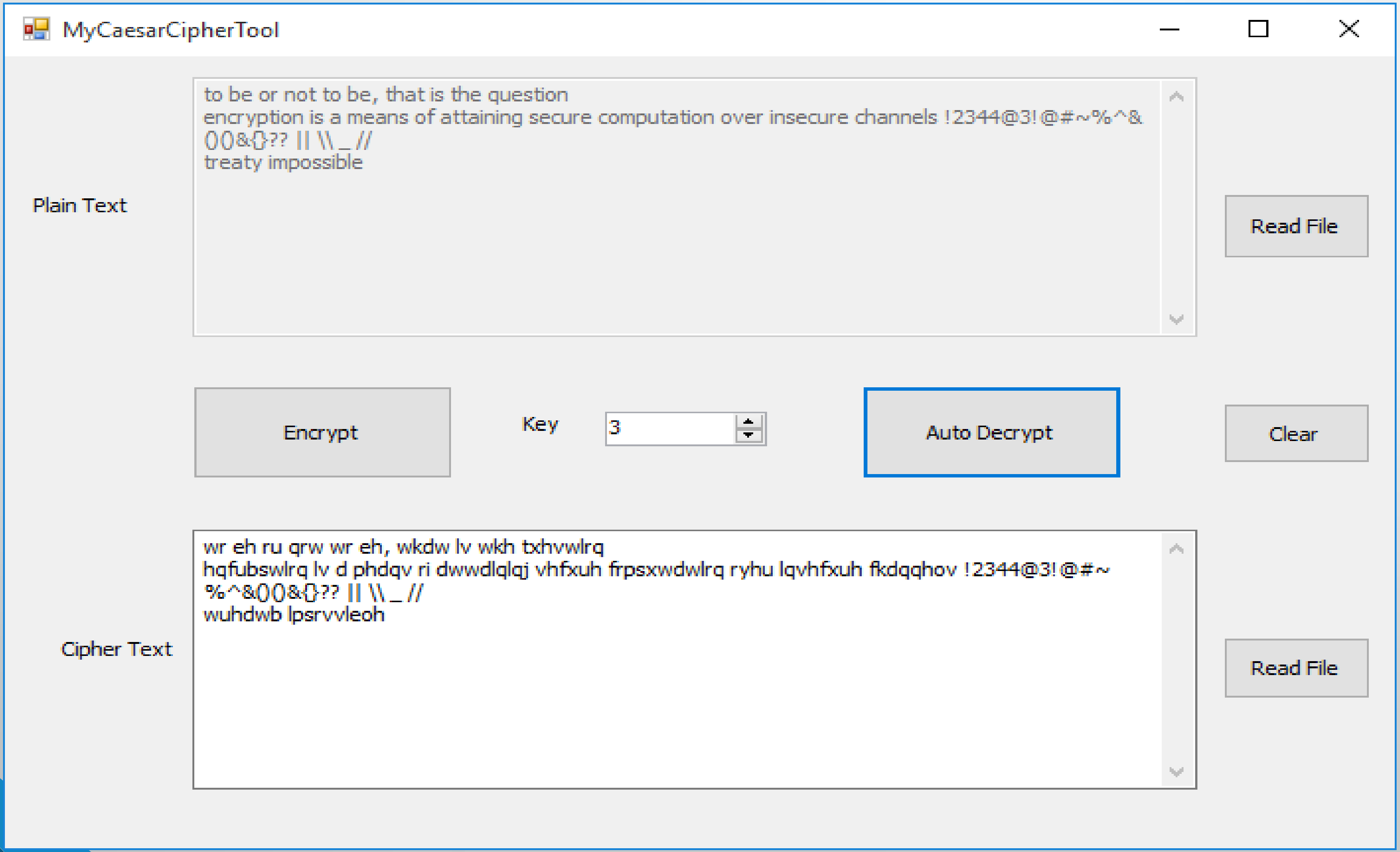
Decryption Function

- Search for each letter of the message in English Dictionary and get its equivalent number
- Subtract letter number with given Key if number was negative then add 26 to it
- take its modulus to 26
- Return equivalent latter for this number
- If message letter not found in dictionary its return same letter such as any special letters (?,!,1,2,…)
- Return Plain message



```
1 reference
private static string Decrypt(string cipherText, int key)
{
    try
    {
        var dic = EnglishLetterDictionary();
        var text = "";

        for (var i = 0; i < cipherText.Length; i++)
        {
            if (!dic.Any(k => k.Key.Equals(cipherText[i])))
            {
                text += cipherText[i];
                continue;
            }
            var tempVal = dic.FirstOrDefault(k => k.Key.Equals(cipherText[i])).Value - key;
            if (tempVal < 0) tempVal += 26;
            tempVal %= 26;
            text += dic.FirstOrDefault(v => v.Value == tempVal).Key;
        }

        return text;
    }
    catch (Exception e)
    {
        return "Error: --> " + e.Message;
    }
}
```

Test Program



```
to be or not to be, that is the question
encryption is a means of attaining secure computation over insecure channels !2344@3!@#~%^&
()&{}?? || \\ _ //
treaty impossible
```

```
wr eh ru qrw wr eh, wkdw lv wkh txhvwlrq
hqfubswlrq lv d phdqv ri dwwdlqlqj vhfxuh frpsxwdwlrq ryhu lqvhfxuh fkdqqhov !2344@3!@#~
%^&()&{}?? || \\ _ //
wuhdwb lpsrvvleoh
```

Thank You