Yasin ÖZTÜRK 704171007

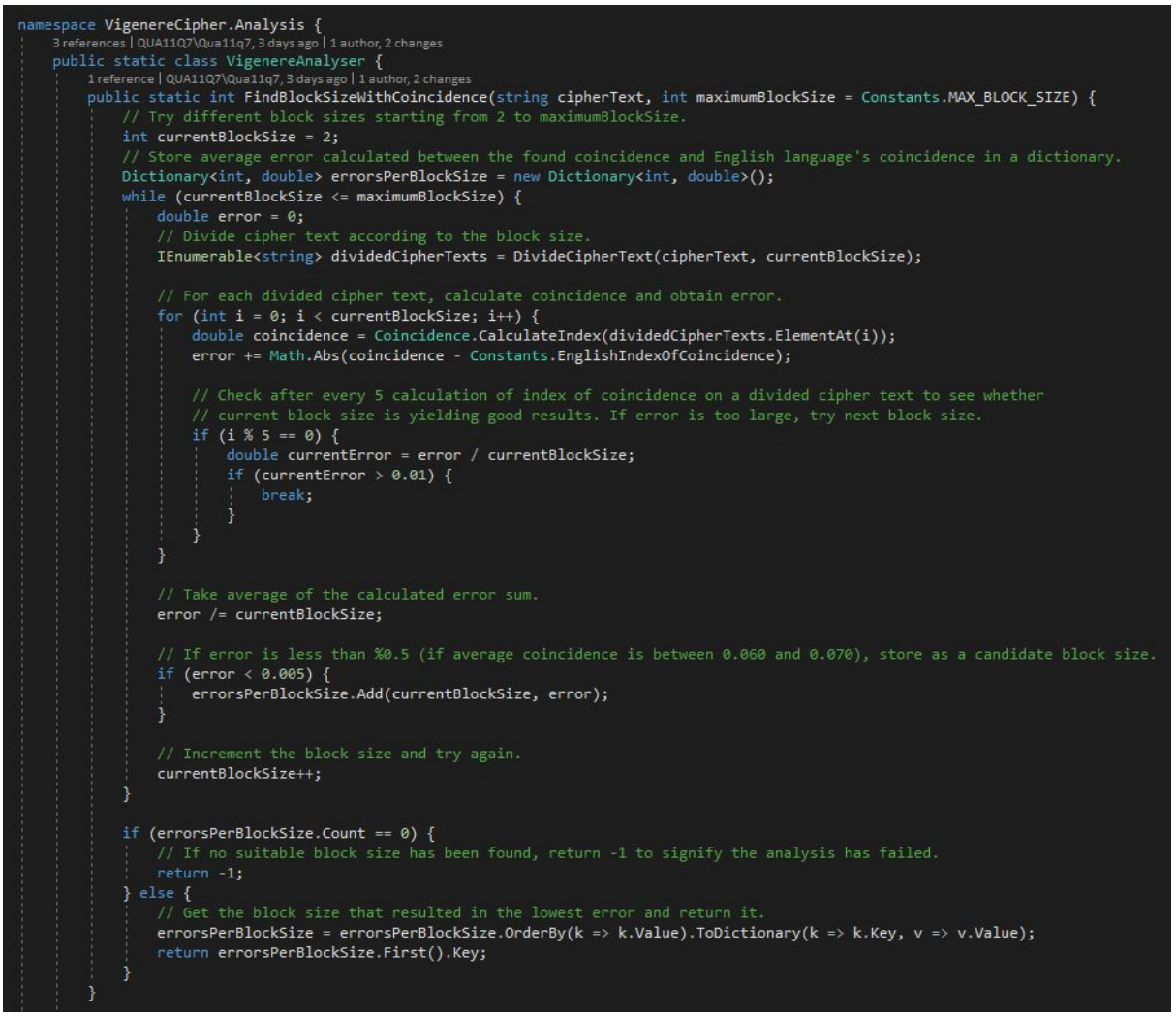
BLG520E Cryptography

1st Homework

1. Vigenere cipher veya bir varyasyonu (Beaufort, Variant) ile şifrelenen metni, şifrelemede kullanılan anahtarı bulmak için öncelikle known ciphertext saldırısı denenmiştir. Known ciphertext attack ile istenen key bulunamadığı için, known plaintext attack işlemi gerçekleştirilmiştir. Aşağıda uygulanan işlemler adım adım anlatılmıştır.

    a. Known ciphertext ile anahtara ulaşmak için Index of Coincidence uygulanmış ve blok uzunluğu bulunmaya çalışılmıştır. Uygulanan Index of Coincidence testinde, blok uzunlukları 2'den 200'e kadar denenmiştir. Uygulanan testte, hiç bir blok uzunluğu, İngilizce'nin sahip olduğu 0.065 indexine 0.005 hata (0.06 - 0.07) ile yaklaşamamıştır. Blok uzunluğu bulunamadığı için known ciphertext attack başarısız olarak sonuçlanmıştır.

```
namespace IndexOfCoincidence {
    1 reference | QUA11Q7\Qua11q7, 3 days ago | 1 author, 1 change
    public static class Coincidence {
        1 reference | QUA11Q7\Qua11q7, 3 days ago | 1 author, 1 change
        public static double CalculateIndex(string text) {
            text = text.ToUpper();

            // Create a dictionary to store each character's occurence.
            Dictionary<char, int> occurences = new Dictionary<char, int> {
                { 'A', 0 }, { 'B', 0 }, { 'C', 0 }, { 'D', 0 }, { 'E', 0 },
                { 'F', 0 }, { 'G', 0 }, { 'H', 0 }, { 'I', 0 }, { 'J', 0 },
                { 'K', 0 }, { 'L', 0 }, { 'M', 0 }, { 'N', 0 }, { 'O', 0 },
                { 'P', 0 }, { 'Q', 0 }, { 'R', 0 }, { 'S', 0 }, { 'T', 0 },
                { 'U', 0 }, { 'V', 0 }, { 'W', 0 }, { 'X', 0 }, { 'Y', 0 }, { 'Z', 0 }
            };

            // Count each characters occurence in the text.
            foreach (char character in text) {
                if (!occurences.ContainsKey(character))
                    occurences[character] = 0;

                occurences[character]++;
            }

            double index = 0;
            int totalLength = occurences.Sum(o => o.Value);
            foreach (KeyValuePair<char, int> occurence in occurences) {
                // Calculate index of coincidence
                double prob = (double)occurence.Value / totalLength;
                index += prob * prob;
            }

            return index;
        }
    }
}
```



```
namespace VigenereCipher.Analysis {
    3 references | QUA11Q7\Qua11q7, 3 days ago | 1 author, 2 changes
    public static class VigenereAnalyser {
        1 reference | QUA11Q7\Qua11q7, 3 days ago | 1 author, 2 changes
        public static int FindBlockSizeWithCoincidence(string cipherText, int maximumBlockSize = Constants.MAX_BLOCK_SIZE) {
            // Try different block sizes starting from 2 to maximumBlockSize.
            int currentBlockSize = 2;
            // Store average error calculated between the found coincidence and English language's coincidence in a dictionary.
            Dictionary<int, double> errorsPerBlockSize = new Dictionary<int, double>();
            while (currentBlockSize <= maximumBlockSize) {
                double error = 0;
                // Divide cipher text according to the block size.
                IEnumerable<string> dividedCipherTexts = DivideCipherText(cipherText, currentBlockSize);

                // For each divided cipher text, calculate coincidence and obtain error.
                for (int i = 0; i < currentBlockSize; i++) {
                    double coincidence = Coincidence.CalculateIndex(dividedCipherTexts.ElementAt(i));
                    error += Math.Abs(coincidence - Constants.EnglishIndexOfCoincidence);

                    // Check after every 5 calculation of index of coincidence on a divided cipher text to see whether
                    // current block size is yielding good results. If error is too large, try next block size.
                    if (i % 5 == 0) {
                        double currentError = error / currentBlockSize;
                        if (currentError > 0.01) {
                            break;
                        }
                    }
                }

                // Take average of the calculated error sum.
                error /= currentBlockSize;

                // If error is less than %0.5 (if average coincidence is between 0.060 and 0.070), store as a candidate block size.
                if (error < 0.005) {
                    errorsPerBlockSize.Add(currentBlockSize, error);
                }

                // Increment the block size and try again.
                currentBlockSize++;
            }

            if (errorsPerBlockSize.Count == 0) {
                // If no suitable block size has been found, return -1 to signify the analysis has failed.
                return -1;
            } else {
                // Get the block size that resulted in the lowest error and return it.
                errorsPerBlockSize = errorsPerBlockSize.OrderBy(k => k.Value).ToDictionary(k => k.Key, v => v.Value);
                return errorsPerBlockSize.First().Key;
            }
        }
```

b. Known ciphertext saldırısı ile anahtar bulunamayınca, known plaintext saldırı gerçekleştirilmiştir. Ciphertext ve plaintext metin halinden sayısal gösterimlerine dönüştürülmüştür. Vigenere cipher'ın tanımından yola çıkarak, K = (C - P) % 26 formülü ile her bir harf için bir kaydırma miktarı bulunmuştur. Bulunan kaydırma miktarları sayı formundan yazıya çevrilmiştir.

c. Bulunan uzun anahtar metninde sürekli tekrar eden substring olup olmadığı kontrol edilmiştir. Bulunacak olan en kısa tekrar eden substring, metnin şifresinde kullanılan anahtarı verecektir. Tekrar eden substring'i bulma işlemi gerçekleştirildiğine, aşağıdaki anahtar metni elde edilmiştir. Elde edilen anahtarın doğruluğunu kontrol etmek adına ciphertext bulunan anahtar ile decrypt edilmiş ve plaintext ile karşılaştırılmıştır.

MPSFWJLOREVIKNQDUHJMPCTGILOBSFHKNAREGJMZQDFILYPCEHKX
OBDGJWNACFIVMZBEHULYADGTKXZCFSJWYBERIVXADQHUWZCPGT
VYBOFSUXANERTWZMDQSVYLCPRUXKBOQTWJANPSVIZMORUHYLNQ
TGXK

```
4 references | QUA11Q7\Qua11q7, 3 days ago | 1 author, 1 change
static bool HomeworkSection1() {
    // Start finding key with only known cipher text attack
    bool successfull = DecryptWithKnownCipherText();

    // If known cipher text attack has not been successfull, try known plain text and cipher text attack.
    if (!successfull) {
        successfull = DecryptWithKnownPlainAndCipherText();
    }

    if (successfull) {
        // Decode the cipher text with the found keyword to check whether the found keyword is correct or not.
        Vigenere vigenere = new Vigenere();
        string decodedText = vigenere.Decode(cipherText, keywordText);

        if (decodedText == plainText)
            Console.WriteLine("Cipher text decrypted successfully!");
    }

    return successfull;
}
```

```
1 reference | QUA11Q7\Qua11q7, 3 days ago | 1 author, 1 change
static bool DecryptWithKnownCipherText() {
    Console.WriteLine("Starting crypto analysis by using only known cipher text.");

    // Convert ciphertext to numbers.
    IEnumerable<int> valuesOfCipherText = Utilities.GetValueArray(cipherText);

    // Apply Index of Coincidence analysis to find out the block size.
    int blockSize = VigenereAnalyser.FindBlockSizeWithCoincidence(cipherText);

    // If block size is -1, return false to signify the known cipher text analysis has failed.
    if (blockSize == -1) {
        Console.WriteLine("Crypto analysis by using only known cipher text failed to obtain block size.");
        return false;
    }

    // Divide cipher texts according to the found block size.
    IEnumerable<string> dividedCipherTexts = VigenereAnalyser.DivideCipherText(cipherText, blockSize);

    // For each divided cipher text, apply frequency analysis to find out the shift amount. Store shift amounts for each divided cipher text.
    int[] shiftAmounts = new int[blockSize];
    for (int i = 0; i < blockSize; i++) {
        int shiftAmount = FrequencyAnalyser.FindShiftAmount(dividedCipherTexts.ElementAt(i));
        // If shift amount is -1, return false to signify the known cipher text analysis has failed.
        if (shiftAmount == -1) {
            Console.WriteLine("Crypto analysis by using only known cipher text failed to obtain shift amount.");
            return false;
        }

        shiftAmounts[i] = shiftAmount;
    }

    // Obtain keyword string from shift values.
    keywordText = Utilities.GetString(shiftAmounts);
    Console.WriteLine($"Cipher key: {keywordText}");

    return true;
}
```

```
1 reference | QUA11Q7\Qua11q7, 3 days ago | 1 author, 2 changes
static bool DecryptWithKnownPlainAndCipherText() {
    Console.WriteLine("Starting crypto analysis by using known plain and cipher text.");

    // Convert plain text and cipher text to numbers.
    IEnumerable<int> valuesOfPlainText = Utilities.GetValueArray(plainText);
    IEnumerable<int> valuesOfCipherText = Utilities.GetValueArray(cipherText);

    // Store all the shift values
    int[] values = new int[plainText.Length];
    for (int i = 0; i < plainText.Length; i++) {
        // Calculate each shift value between plain text and cipher text according to Vigenere cipher.
        values[i] = ((valuesOfCipherText.ElementAt(i) + 26) - valuesOfPlainText.ElementAt(i)) % 26;
    }

    // Create the long keyword text from obtained shift values.
    string longKeyword = Utilities.GetString(values);

    // Try to figure out the shortest key possible, i.e find out the repeating text in the long keyword text.
    int blockSize = 1;
    while (blockSize++ <= Constants.MAX_BLOCK_SIZE) {
        List<string> dividedKeyword = VigenereAnalyser.DivideCipherText(longKeyword, blockSize);

        // If a divided keywords all characters are the same, break out of the loop.
        bool allSame = dividedKeyword.TrueForAll(k => k.Distinct().Count() == 1);
        if (allSame)
            break;
    }

    if (blockSize >= Constants.MAX_BLOCK_SIZE) {
        // Found keyword's size is equal to the size of the plain text, obtained keyword cannot be correct.
        Console.WriteLine("Found cipher key's length is equal to the cipher text's length. Cipher key couldn't be found!");
        return false;
    } else {
        // Get the actual keyword.
        keywordText = longKeyword.Substring(0, blockSize);
        Console.WriteLine($"Cipher key: {keywordText}");

        return true;
    }
}
```

2. İlk bölümde uzunluğu 156 karakter olarak bulunan anahtar metni ile eş uzunlukta rastgele bir anahtar daha oluşturulmuştur. Rastgele oluşturulan anahtar ile, verilen ciphertext Vigenere cipher kullanılarak şifrelenmiştir. Elde edilen ciphertext ve plaintext'e, ilk bölümdeki aşamalar uygulanarak anahtar metin elde edilmeye çalışılmıştır. İlk bölümde olduğu gibi, known ciphertext saldırısı anahtarı bulamazken, known plaintext saldırısı ile kompozit anahtar elde edilmiştir. Aşağıda uygulanan işlemler adım adım anlatılmıştır.

    a. Rastgele oluşturulacak bir anahtar ile şifreleme yapılacağı için, Vigenere cipher'ın şifreleme ve şifre çözme methodları geliştirilmiştir.

    b. Rastgele oluşturulan anahtar ile şifreleme işlemi gerçekleştirilmiştir. Elde edilen ciphertext'e Index of Coincidence işlemi uygulanmıştır. İşlem sonucunda herhangi bir blok uzunluğu elde edilememiştir.

```
3 references | QUA11Q7\Qua11q7, 3 days ago | 1 author, 2 changes
public abstract class BaseVigenere {
    1 reference | QUA11Q7\Qua11q7, 3 days ago | 1 author, 2 changes
    public string Encode(string plainText, string keyword) {
        // Convert text values to corresponding integer values.
        IEnumerable<int> textValues = Utilities.GetValueArray(plainText);
        IEnumerable<int> keywordValues = Utilities.GetValueArray(keyword);

        // Obtain cipher values from appling an operation between text values and keyword value.
        // Different ciphers apply different operations.
        // Vigenere: C = (P + K) % 26
        // Beaufort: C = (K - P) % 26
        // Variant:  C = (P - K) % 26
        int[] cipherValues = new int[plainText.Length];
        for (int i = 0; i < plainText.Length; i++) {
            cipherValues[i] = CalculateEncodeValue(textValues.ElementAt(i), keywordValues.ElementAt(i % keyword.Length));
        }

        // Convert cipher values to text.
        return Utilities.GetString(cipherValues);
    }

    1 reference | QUA11Q7\Qua11q7, 3 days ago | 1 author, 2 changes
    public string Decode(string cipherText, string keyword) {
        // Convert text values to corresponding integer values.
        IEnumerable<int> textValues = Utilities.GetValueArray(cipherText);
        IEnumerable<int> keywordValues = Utilities.GetValueArray(keyword);

        // Obtain plain text values from appling an operation between cipher values and keyword value.
        // Different ciphers apply different operations.
        // Vigenere: P = (C - K) % 26
        // Beaufort: P = (K - C) % 26
        // Variant:  P = (C + K) % 26
        int[] plainTextValues = new int[cipherText.Length];
        for (int i = 0; i < cipherText.Length; i++) {
            plainTextValues[i] = CalculateDecodeValue(textValues.ElementAt(i), keywordValues.ElementAt(i % keyword.Length));
        }

        // Convert plain text values to text.
        return Utilities.GetString(plainTextValues);
    }

    // These abstract functions are implemented in Vigenere, Beaufort and Variant classes.
    4 references | QUA11Q7\Qua11q7, 3 days ago | 1 author, 2 changes
    protected abstract int CalculateEncodeValue(int plainTextValue, int keywordValue);
    4 references | QUA11Q7\Qua11q7, 3 days ago | 1 author, 1 change
    protected abstract int CalculateDecodeValue(int cipherTextValue, int keywordValue);
}
```

c. Known ciphertext saldırısı başarısız sonuçlandığı için, known plaintext saldırısı gerçekleştirilmiştir. Saldırı sonucunda aşağıdaki anahtar metni elde edilmiştir. Elde edilen anahtarın doğruluğunu kontrol etmek adına ciphertext bulunan anahtar ile decrypt edilmiş ve plaintext ile karşılaştırılmıştır.

Rastgele Anahtar:
VNFXNBYCLLAIBSYHVZBAFHAGIMOIUTKMAVDHQCYBCRFBELHPYTPU
OBOOMYOBMQIRBOIFPRQSBZXNCZCXEZJAEPXYHPUROXOMJNZQDB
AGCWTOEALNWRWNGHNLRKYZXZEDLKEOZLTCZCXHWITGDGUWABF
CAOZS

Kompozit Anahtar:
HCXCJKJQCPVQLFOKPGKMUJTMQXCJMYRWNVULWLKASUKJPJWRCA
ZRCCRUVUBBOVQMNNJJWLBQBCDGMWBZJRSWCQBPPKRRRNVGFM
BFJUVEDKYGYXLAAIPJFTQBJFWKZOVXIUFCPEPLZPMZRQSSRXODYM
SSTUWC

```
1 reference | QUA11Q7\Qua11q7, 3 days ago | 1 author, 1 change
static void HomeworkSection2() {
    // Generate random key with the same length as the keyword.
    randomKeyword = Utilities.GenerateRandomKey(keywordText.Length);
    Console.WriteLine($"Selected Random Key: {randomKeyword}");

    // Encrypt cipher text with the random key.
    Vigenere vigenere = new Vigenere();
    cipherText = vigenere.Encode(cipherText, randomKeyword);

    // Apply same crypto analysis methods in section 1.
    HomeworkSection1();
}
```

3. Üçüncü bölümde, permutation cipher'da kullanılmak üzere rastgele bir anahtar seçilmiştir. Permutation cipher'da crypto analiz uygulamak uzun zaman alabileceğinden, seçilen rastgele anahtarın uzunluğu 7 karakter olarak belirlenmiştir. Ciphertext permutation cipher ile son üretilen rastgele anahtar kullanılarak şifrelenmiştir. Şifreleme sonrasında, birinci bölümde uygulanan known ciphertext ve known plaintext saldırıları uygulanmıştır. Uygulanan saldırılarda, permutation cipher için herhangi bir crypto analiz yöntemi uygulanmamıştır. Uygulanan known ciphertext saldırısı sonuç vermerken, known plaintext saldırısı uzunluğu metnin uzunluğuna eşit bir anahtar metni döndürmüştür. Döndürülen anahtar metni kullanışlı olmayacak kadar uzun olduğundan, bu crypto analiz yöntemi başarısızlıkla sonuçlanmıştır. Birinci bölümde uygulanan yöntemler uygulanmadan önce, permutation cipher'ı kırmak üzere, uzunluğu 2'den başlayarak sırayla permütasyonlar üretilmiştir. Üretilen permütasyon ile cipher decrypt edildikten sonra birinci bölümde uygulanan işlemler tekrarlanmıştır. Birçok denemeden sonra doğru permütasyon anahtarı bulunmuş ve known plaintext saldırısı ile Vigenere cipher anahtarı elde edilmiştir. Aşağıda uygulanan işlemler adım adım anlatılmıştır.

    a. Rastgele oluşturulacak bir anahtar ile şifreleme yapılacağı için, Permutation cipher'ın şifreleme ve şifre çözme metodları geliştirilmiştir.

    b. Rastgele oluşturulan anahtar ile şifreleme işlemi gerçekleştirilmiştir. Elde edilen ciphertext'e bölüm 1'de bahsedilen işlemler uygulanmıştır. İşlem sonucunda herhangi bir anahtar elde edilememiştir.

    c. Permütasyon şifrelemesini kırmak üzere, uzunluğu 2'den başlayarak tüm permütasyonlar üretilerek mevcut cipher text decode edilmiştir. Decode sonucu elde edilen metne, bölüm 1'deki testler uygulanmıştır. Bölüm 1'deki testler yanlış denenen her permütasyon için doğru sonuç üretememiştir. Bölüm 1'de kullanılan metodlar doğru sonuç ürettiğinde, denenen permütasyon anahtarının, rastgele üretilen permütasyon anahtarı ile aynı olduğu gözlemlenmiştir. Aşağıda rastgele seçilen permütasyon anahtarı ve permütasyonu decode ederken kullanılan doğru permütasyon anahtarı gösterilmiştir:

Rastgele Anahtar:
RTVFNVT => {17, 19, 21, 5, 13, 21, 19} => {3, 4, 0, 1, 6, 2, 5}

Deneme Sonucu Elde Edilip Doğru Sonuç Veren Permütasyon Anahtarı:
{3, 4, 0, 1, 6, 2, 5}

```
1 reference | QUA11Q7\Qua11q7, 1 day ago | 1 author, 2 changes
static void HomeworkSection3() {
    // Generate random key to apply permutation cipher. The key size is kept relatively small for crypto analysis.
    permutationKeyword = Utilities.GenerateRandomKey(7);

    // Encrypt the cypertext by using permutation cipher with random permutation key.
    Permutation permutation = new Permutation();
    cipherText = permutation.Encode(cipherText, permutationKeyword);

    // Apply same crypto analysis methods in section 1.
    bool successfull = HomeworkSection1();

    // If crypto analysis has failed, try decoding the permutation cipher first, then try steps defined in Section1.
    if (!successfull) {
        DecryptPermutationCypher();
    }
}
```

```
1 reference | QUA11Q7\Qua11q7, 3 days ago | 1 author, 1 change
static bool DecryptPermutationCypher() {
    // Create a Permutation class for encoding and decoding texts.
    Permutation permutationCipher = new Permutation();

    // Store original cipher text.
    string originalCipherText = cipherText;

    // Start trying permutations from length 2.
    int currentPermutationLength = 2;
    while (currentPermutationLength <= Constants.MAX_PERMUTATION_LENGTH) {
        // Create a list containing numbers to permutate, starting from 0 to currentPermutationLength.
        int[] list = new int[currentPermutationLength];
        for (int i = 0; i < currentPermutationLength; i++) {
            list[i] = i;
        }

        // Get all permutations of the list.
        IEnumerable<IEnumerable<int>> permutations = GetPermutations(list, currentPermutationLength);

        // Decode cipher text with each possible permutations. Apply crypto analysis defined in Section1 to decoded cipher text.
        foreach (IEnumerable<int> permutation in permutations) {
            cipherText = permutationCipher.Decode(originalCipherText, permutation);

            // Apply crypto analysis for obtained decoded cipher.
            bool successfull = HomeworkSection1();

            // If crypto analysis has been successfull, current permutation is the correct key for the permutation cipher.
            if (successfull) {
                Console.WriteLine($"Permutation cipher has been decrypted successfully." +
                    $" Permutation Cipher Key: {GetPermutationString(permutation)}");
                return true;
            }
        }

        currentPermutationLength++;
    }

    return false;
}
```

Ödevde kullanılan tüm kodlar github’a yüklenmiştir: [qua11q7/Cryptography-Homework1]