Applied Cryptography

Assignment-1

Name:-Aayush Arora

Roll Number:- 2110110017

Problem1: Implement the attacks described for the shift cipher and the Vigenère cipher. Your code should take a ciphertext as input and output the key alongwith the plaintext.

Approach: Breaking a Shift Cipher

1. **Input Ciphertext:**
   - The user provides the Shift cipher ciphertext as input.
2. **Data Preprocessing:**
   - Convert the input ciphertext to uppercase to ensure uniformity.
   - Create a map of English letter frequencies to assist in deciphering the text.
3. **Calculate Letter Frequencies in Ciphertext:**
   - Initialize a map to store the frequencies of each letter in the ciphertext.
   - Iterate through the ciphertext and count the occurrences of each letter while ignoring non-alphabetic characters.
   - Keep track of the total number of letters in the ciphertext.
4. **Calculate Ij Values for All Shifts:**
   - For each possible shift value (0 to 25), calculate Ij values.
   - Ij values represent the expected frequency distribution of letters in the decrypted text, assuming a particular shift.
   - For each shift, iterate through the map of English letter frequencies and calculate the expected frequency of each letter in the decrypted text.
   - Sum up the products of the expected frequencies and observed frequencies for all letters.
   - This step helps in identifying the shift that results in a distribution closest to typical English letter frequencies.
5. **Select the Best Shift Key:**
   - Set a target Ij value based on typical English letter frequency (e.g., 0.065 for the letter 'E').
   - Compare the calculated Ij values for each shift with the target Ij value.

- Select the shift with the Ij value closest to the target.
- This shift is likely the one used in the encryption and serves as the decryption key.

6. **Decrypt the Ciphertext:**
   - Use the determined shift key to decrypt the ciphertext.
   - For each character in the ciphertext, apply the reverse shift operation to obtain the original plaintext character.
   - Non-alphabetic characters remain unchanged.
   - Build the decrypted plaintext character by character.
7. **Output the Result:**
   - Display the found shift key.
   - Display the decrypted plaintext, which is the result of breaking the Shift cipher.
8. **Conclusion:**
   - The program provides the user with the ability to break a Shift cipher by analyzing the frequency distribution of letters and comparing it to expected English letter frequencies.
   - The result is a decrypted message, revealing the original plaintext.

Note:

- The accuracy of the decryption depends on the quality of the ciphertext and the availability of sufficient text for statistical analysis.
- In practice, this method may not work well for very short or highly encrypted texts.

This approach leverages statistical analysis and the known characteristics of the English language to identify the most likely shift key used in the Shift cipher, ultimately allowing for the decryption of the ciphertext.

CODE

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

class EnglishLetterFrequencies {

    public static final Map<Character, Double> letterFrequencies = new
HashMap<>();
```

```java
    static {
        letterFrequencies.put('A', 0.08167);
        letterFrequencies.put('B', 0.01492);
        letterFrequencies.put('C', 0.02782);
        letterFrequencies.put('D', 0.04253);
        letterFrequencies.put('E', 0.12702);
        letterFrequencies.put('F', 0.02228);
        letterFrequencies.put('G', 0.02015);
        letterFrequencies.put('H', 0.06094);
        letterFrequencies.put('I', 0.06966);
        letterFrequencies.put('J', 0.00153);
        letterFrequencies.put('K', 0.00772);
        letterFrequencies.put('L', 0.04025);
        letterFrequencies.put('M', 0.02406);
        letterFrequencies.put('N', 0.06749);
        letterFrequencies.put('O', 0.07507);
        letterFrequencies.put('P', 0.01929);
        letterFrequencies.put('Q', 0.00095);
        letterFrequencies.put('R', 0.05987);
        letterFrequencies.put('S', 0.06327);
        letterFrequencies.put('T', 0.09056);
        letterFrequencies.put('U', 0.02758);
        letterFrequencies.put('V', 0.00978);
        letterFrequencies.put('W', 0.02360);
        letterFrequencies.put('X', 0.00150);
        letterFrequencies.put('Y', 0.01974);
        letterFrequencies.put('Z', 0.00074);
    }
}

public class CipherCracker {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the Shift cipher ciphertext: ");
        String ciphertext = scanner.nextLine().toUpperCase();

        System.out.println("Attempting to break the Shift cipher...");
        int key = breakShiftCipher(ciphertext);

        if (key != -1) {
            System.out.println("Found key: " + key);
            String plaintext = decryptShiftCipher(ciphertext, key);
            System.out.println("Decrypted text: " + plaintext);
        } else {
            System.out.println("Failed to break the Shift cipher.");
        }
```

```
        scanner.close();
    }

    public static int breakShiftCipher(String ciphertext) {
        // Calculate letter frequencies in the ciphertext
        Map<Character, Integer> letterFrequencies = new HashMap<>();
        int totalLetters = 0;

        for (char c : ciphertext.toCharArray()) {
            if (Character.isLetter(c)) {
                char uppercaseChar = Character.toUpperCase(c);
                letterFrequencies.put(uppercaseChar,
letterFrequencies.getOrDefault(uppercaseChar, 0) + 1);
                totalLetters++;
            }
        }

        // Calculate Ij values for all shifts (0 to 25)
        double[] ijValues = new double[26];

        for (int j = 0; j < 26; j++) {
            for (char pi :
EnglishLetterFrequencies.letterFrequencies.keySet()) {
                char qi = (char)  + (pi  + j) ;
                double pij = (double) letterFrequencies.getOrDefault(qi, 0) /
totalLetters;
                ijValues[j] +=
EnglishLetterFrequencies.letterFrequencies.get(pi) * pij;
            }
        }

        // Find the key with the closest Ij value to the expected value
        double targetIj = 0.065;
        int bestKey = -1;
        double closestIj = Double.MAX_VALUE;

        for (int key = 0; key < 26; key++) {
            double diff = Math.abs(ijValues[key] - targetIj);
            if (diff < closestIj) {
                closestIj = diff;
                bestKey = key;
            }
        }

        return bestKey;
    }
```

```java
    public static String decryptShiftCipher(String ciphertext, int shift) {
        StringBuilder decryptedText = new StringBuilder();

        for (char c : ciphertext.toCharArray()) {
            if (Character.isLetter(c)) {
                char decryptedChar = (char) ('A' + (c - 'A' - shift + 26) %
26);
                decryptedText.append(decryptedChar);
            } else {
                decryptedText.append(c);
            }
        }

        return decryptedText.toString();
    }
}
```

OUTPUT

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\aayus\Desktop\Sem-5\Applied Cryptography\Assignments>  & 'C:\Program Files\Java\jdk-19\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\
Users\aayus\AppData\Roaming\Code\User\workspaceStorage\7807b792ea56be5f5964aa7b62aba17f\redhat.java\jdt_ws\Assignments_61eefebc\bin' 'CipherCracker'
Enter the Shift cipher ciphertext: Nyllulyf, pu hss paz sbzoulzz huk cpiyhujf, pz hu pualnyhs whya vm vby uhabyhs lucpyvutlua. Pa lujvtwhzzlz aol tfyphk vm ayllz, zo
ybiz, wshuaz, huk nyhzzlz aoha ishurla vby shukzjhwlz, wyvcpkpun tvyl aohu qbza hlzaolapj hwwlhs. Pa wshfz h wpcvahs yvsl pu vby spclz, pumsblujpun vby wofzpjhs olhs
ao, tluahs dlss-ilpun, huk aol zbzahpuhipspaf vm vby wshula.Ha paz jvyl, nyllulyf pz h zftivs vm spml huk cpahspaf. Pa pz aol spcpun jhuchz aoha whpuaz aol dvysk hyv
buk bz. Myvt aol avdlypun mvylzaz aoha dopzwly zljylaz vm aptlslzzulzz av aol klspjhal wlahsz vm h zwypun isvzzvt, nyllulyf jhwabylz aol lzzlujl vm nyvdao, yluldhs,
huk ohytvuf dpao uhabyl. Fla, paz zpnupmpjhujl nvlz mhy ilfvuk tlyl hlzaolapjzVul vm aol tvza lcpklua hkchuahnlz vm nyllulyf pz paz wvzpapcl ptwhja vu aol lucpyvutlu
a. Ayllz huk wshuaz hja hz uhabyhs hpy wbypmplyz, hizvyipun ohytmbs wvssbahuaz zbjo hz jhyivu kpvepkl huk ylslhzpun vefnlu. Aolf tpapnhal aol lmmljaz vm jspthal johu
nl if ylkbjpun nylluovbzl nhz ltpzzpvuz huk zahipspgpun altwlyhabylz. Tvylvcly, aol yvva zfzaltz vm wshuaz olsw wylclua zvps lyvzpvu, luzbypun aol svun-alyt mlyapspa
f vm aol shukNyllulyf hszv wshfz h jybjphs yvsl pu ipvkpclyzpaf. Kpclyzl ljvzfzaltz aoypcl dolyl aolyl pz hu hibukhujl vm wshua spml, wyvcpkpun ohipahaz mvy jvbuaslz
z zwljplz vm hupthsz huk puzljaz. Aopz pualyjvuuljalk dli vm spml luzbylz aol ihshujl vm uhabyl huk aol zbycpchs vm thuf zwljplz, pujsbkpun vby vdu
Attempting to break the Shift cipher...
Found key: 7
Decrypted text: GREENERY, IN ALL ITS LUSHNESS AND VIBRANCY, IS AN INTEGRAL PART OF OUR NATURAL ENVIRONMENT. IT ENCOMPASSES THE MYRIAD OF TREES, SHRUBS, PLANTS, AND G
RASSES THAT BLANKET OUR LANDSCAPES, PROVIDING MORE THAN JUST AESTHETIC APPEAL. IT PLAYS A PIVOTAL ROLE IN OUR LIVES, INFLUENCING OUR PHYSICAL HEALTH, MENTAL WELL-BEI
NG, AND THE SUSTAINABILITY OF OUR PLANET.AT ITS CORE, GREENERY IS A SYMBOL OF LIFE AND VITALITY. IT IS THE LIVING CANVAS THAT PAINTS THE WORLD AROUND US. FROM THE TO
WERING FORESTS THAT WHISPER SECRETS OF TIMELESSNESS TO THE DELICATE PETALS OF A SPRING BLOSSOM, GREENERY CAPTURES THE ESSENCE OF GROWTH, RENEWAL, AND HARMONY WITH NA
TURE. YET, ITS SIGNIFICANCE GOES FAR BEYOND MERE AESTHETICSONE OF THE MOST EVIDENT ADVANTAGES OF GREENERY IS ITS POSITIVE IMPACT ON THE ENVIRONMENT. TREES AND PLANTS
 ACT AS NATURAL AIR PURIFIERS, ABSORBING HARMFUL POLLUTANTS SUCH AS CARBON DIOXIDE AND RELEASING OXYGEN. THEY MITIGATE THE EFFECTS OF CLIMATE CHANGE BY REDUCING GREE
NHOUSE GAS EMISSIONS AND STABILIZING TEMPERATURES. MOREOVER, THE ROOT SYSTEMS OF PLANTS HELP PREVENT SOIL EROSION, ENSURING THE LONG-TERM FERTILITY OF THE LANDGREENE
RY ALSO PLAYS A CRUCIAL ROLE IN BIODIVERSITY. DIVERSE ECOSYSTEMS THRIVE WHERE THERE IS AN ABUNDANCE OF PLANT LIFE, PROVIDING HABITATS FOR COUNTLESS SPECIES OF ANIMAL
S AND INSECTS. THIS INTERCONNECTED WEB OF LIFE ENSURES THE BALANCE OF NATURE AND THE SURVIVAL OF MANY SPECIES, INCLUDING OUR OWN
PS C:\Users\aayus\Desktop\Sem-5\Applied Cryptography\Assignments> 
```

VIGENERE CIPHER

Approach Documentation for Breaking a Vigenère Cipher:

Step 1: Find the Most Frequent Trigram

- First, we look for groups of three consecutive letters (trigrams) in the ciphertext.

- We count how many times each trigram appears in the text.

Step 2: Find the Maximum Occurring Trigram

- We identify the trigram that occurs most frequently in the ciphertext. This is a hint at the key length.

Step 3: Calculate Starting Indexes of the Trigrams

- We find the positions in the ciphertext where the most frequent trigram starts. This helps us find the distances between them.

Step 4: Find the Key Length

- We calculate the Greatest Common Divisor (GCD) of the distances between the most frequent trigram occurrences. This GCD is likely to be the key length.

Step 5: Find the Key

- We break the ciphertext into segments equal to the key length.
- For each segment, we try to find the key for that part using a method called "Index of Coincidence."
- We combine the keys from each segment to form the final key used for decryption.

Step 6: Decrypt the Vigenère Cipher

- With the key in hand, we decrypt the ciphertext using the Vigenère cipher decryption method.
- We apply the reverse shift operation for each character in the ciphertext to obtain the original plaintext.

Step 7: Display the Result

- We display the decrypted text, which is the result of breaking the Vigenère cipher.
- The key length and the found key are also displayed.

Note:

- This approach uses statistical analysis and mathematical techniques to break the Vigenère cipher.
- It assumes that the ciphertext is long enough for accurate analysis and that the key length is not extremely long.

- The accuracy of the decryption depends on the quality of the ciphertext and the key length.

CODE

```
import java.util.*;

class EnglishLetterFrequencies {

    public static final Map<Character, Double> letterFrequencies = new
HashMap<>();

    static {
        letterFrequencies.put('a', 0.082);
        letterFrequencies.put('b', 0.015);
        letterFrequencies.put('c', 0.028);
        letterFrequencies.put('d', 0.043);
        letterFrequencies.put('e', 0.13);
        letterFrequencies.put('f', 0.022);
        letterFrequencies.put('g', 0.02);
        letterFrequencies.put('h', 0.061);
        letterFrequencies.put('i', 0.07);
        letterFrequencies.put('j', 0.002);
        letterFrequencies.put('k', 0.008);
        letterFrequencies.put('l', 0.04);
        letterFrequencies.put('m', 0.024);
        letterFrequencies.put('n', 0.067);
        letterFrequencies.put('o', 0.075);
        letterFrequencies.put('p', 0.019);
        letterFrequencies.put('q', 0.001);
        letterFrequencies.put('r', 0.06);
        letterFrequencies.put('s', 0.063);
        letterFrequencies.put('t', 0.091);
        letterFrequencies.put('u', 0.028);
        letterFrequencies.put('v', 0.01);
        letterFrequencies.put('w', 0.024);
        letterFrequencies.put('x', 0.002);
        letterFrequencies.put('y', 0.02);
        letterFrequencies.put('z', 0.001);
    }
}



public class VigenereCipherCracker {
    public static void main(String[] args) {
```

```java
        String ciphertext =
"IFSPPSFJKBOWNWHDNIGSPSGDCBRGKPFLPQMTUOBTPHSRTOZACFHZHCICPOHFTOZPPJWCQBAPPHWEG
BQZODODUSGEJSAJTWOOQTHCGSGDJFIMUDZLPHGLPRUCCGGPUHVLVPZLPYSEQIFWCBRDEODPUDFZXWR
TPUAZTSHSCBXFUHOPUHVPVWQLRDSLNWHANOMDCDWGQHOWTCZPKBCFTZWGGGWYHZIPPQWYICICRVMDK
QOWJSOWVVAPPHOWYSZWDSWYIOBOVVSDWGHLKBOMKZWEACTZWFDWCBSECHWEUQCCGUFPGBSCAWGLUMA
MQZCQNWTPCBRGKHOWKHMTVWGEJSZTXWBREOBGCGHSCHDLKBHDVVSHQFZOCFCFPRIDHFCXVVSEQKSCK
BUQQFSDVGHSCHKSKGDPTGSNTSHDQTHTOSZPUGBPUGHZVVSOGZWNCHSAGHOWUCTLUDFTPUPWQGGZOUF
PGBSCAQOAVIFPUHVPGGGPPQSZHUFZYHVCGBSHCZOYFVOCOCBJYWHSPOHFTSMPVWHDUWUYKTWNCBQPI
CSDHOFMGMCYFASCGOSDVVSEKQGZPSCQVVSXQGHPXWRPPHOOXOBECUSDQTUCGSBPTMWDKHGAQGWEKJS
TODONVCBEJSSYXWFZPASYVHFPGGOYFDZLPHGLEHODPOHFTOZLKFDFTWTTGFGLDGCCDWBRJOFXHIZAQ
ZZFVOBEUGINJOGNCFPZPRWZZWRPCBRCGZSLUWBRQLMRGBHSGMATVWULVSHSGSTQGQHDQTQWKAOEGQV
LPUSMAFSOWQWYIUFPGBVZWGSRCGSXKGGTQBGLPRGECPWWKNWYIHSXRSFLVIFPUACCGCJPTHVPTCCEU
MGEGAGZHDZLPHGSGZDATSJPPHGZKZSCQGWZPSBDWFWYIHVPNCBRVSFXHSFEKZWEACTEJSZLPRUCGSB
PTMOWUCDWCMGLEFINKOZCQZSTPPWZFWJPTGWEARWGGFGPGQCDAGHPOGHSTWJPYVSCGHVPTSWDCBOMW
BRLPQSZHDZLPHZTHSDCQJWOKBUSCPWECHGQQFQZWBHWGGGDRSQTGGCQCBWXCZGLPRWYUSQEUHVTUWB
EGFQZPBSNVSRHGPCQNWTPGBGFTSGEJSPLNOBNGCTYCHICGOBOVVSDWFJTXOZZHAOYAGDPEWSDKBQWW
RWYICICQKBTPIFMCBOCGOGRTSSYGFMECYSDQBOOFSRDKUBTHWQLPQSACFYDCBRRTSSYUDONGGDCQJW
OGOFPUDWEGTFZOHVPECBNTSHPLIBRNSCQHSFTPUDPQDZPCDZLESHZTSZLZSLPTQWDGOBOECBYGQHHK
HVYCHICGOQNGGGEQUFPGBOCGOGHKHVTPQWEKSGSCGPPGBZTPYSOVCWXRFCGGRAPPHOWJSOWVVFPFIQ
PFGHCGGGWGJSWUOBOKBQCGOGPFCJPTOZWYSZWDSWYIOGFTPOYKNOEKCBNQBHTPISDVCFTUSDCGGSCX
WBRCBRPZDOYFWBRIFSPPGDLESGTUSGDGBHTCZTZTHVPJSOWVVOYFVOARWBPUGCQEWHJFKSWNSFDHIF
EJSFXQFSRTSSYGFMSCGSNQBCXKQPPPSTTVGHSGOUCKQIWVIFLNWBOWGHCAFSWKSGSGOJTNMCYVVSNW
ZHTXOHTQBCQEFCAUOBOVVSNCFSZHZWGGGHZEYKSKQVOGDSYFCBQGFHTNSGZKZOYFOVZUDWECPZPEZW
XCHSRTSSYGFMSGZDDOOWYVOWYVVSDGQCYFWHTQBGPPGICKBULUHOMNSTZQRGFRDZJCBRDWDDZTHWYI
ZWGGZWSQCRDVVSTODCCVOBNGCTRTSSYGFMNCBBZVPSZXSFDVOHPFKVPPWHNQASDVCSOWQOEKCBLPRO
HCFSYGGGTVGSCXSGLUOZTXWBREZODUFCZOHSLEVWYIIGLDCIEVVSNAQZPUCTWKTSPECZZIMOYFHVPF
SZTEOHPDOZLPQSZHSQZUMGEGAGRTSSYUDONGGCQHSFZRDCCVIBTVWSDHCFNJWZOTSBEQQCYPSQEYWH
SPOHFTSTZUHSCKBULUSBDGCTCGGDZPGWMKZWEAOBOUHSHCFRDJWDQQFHSGSBGKFCYOSBEHFCXCBSLT
ZMLISVZYSJPTRSDRWHPKHGXCBMLFJOYVOUPUUFPGBSCATONGGHSTSOEUWBEJSTZTACQFSTZTSGECHW
ZPVOMKHOEFSGETIQEKCBLPRDZNZIEKCBLUGHPYOFOUCTEJWGANOBPVWHTUCICTSGAQBGTDWZTVMHZR
FCEGQHLPRDCGGSCXSHSGUFPGBSCAHVLVGIDVOWYUIGTPWHTCHWGGGGFEVODCTTZTSGECHWZPFSQQFS
DVOHTQBOYFGIDVOWYCPZPNOBOOOBLISAPPHOCGQFFEWOWUHSAUHCHCFRPPGICKBUEJSQZPHWYWSRGK
HOWKHMZHCICIFSPPGDLESGTPQCYEZIDKCBRTSSYGFMTUBCEOSFPNMOYGAPPNZWDJASYVWBEJSHLRSG
ETMCQQIFHQFZOKHWDVVSEJFSLFHVLVPWYFGIDVCHSGBOEWFOWYCFWFOBOUIGECWBDQIFPZWGEGBQPH
FCXVVSLKFKPDFSLVVSEQHVPOSBECZGZNOQPKHDCQJWOGGUCGSBPTMSYTWQSGGCFTZWGGGWYECIYVZS
DUKOJUOGHGACGGTCCYOFONSHFUQSWGPFLVSDCQHSNVOBOEIZEKJOEGUFPGBSCAFSNQUBTBWBRKHGAT
CTZWBRTODCCVOBNGHCZWFVPCZHSJODAKBSDUOBOVVSQWHICGCTZWFDWCBSE";
       ciphertext=ciphertext.toLowerCase();
        int keyLength = findKeyLength(ciphertext);
        System.out.println("Key length is "+keyLength);
        String key = findKey(ciphertext, keyLength);
        System.out.println("key is  "+key);
        String plaintext = decryptVigenere(ciphertext, key);
        System.out.println("Decrypted Text: " + plaintext);
    }

    // Step 1: Find the most frequent trigram and store it in a HashMap
```

```
    public static HashMap<String, Integer> findMostFrequentTrigram(String
ciphertext) {
        HashMap<String, Integer> trigramMap = new HashMap<>();
        for (int i = 0; i < ciphertext.length() - 2; i++) {
            String trigram = ciphertext.substring(i, i + 3);
            trigramMap.put(trigram, trigramMap.getOrDefault(trigram, 0) + 1);
        }
        return trigramMap;
    }

    // Step 2: Find the maximum occurring trigram
    public static String findMaximumOccurringTrigram(HashMap<String, Integer>
trigramMap) {
        return Collections.max(trigramMap.entrySet(),
Map.Entry.comparingByValue()).getKey();
    }

    // Step 3: Calculate starting indexes of the trigrams
    public static ArrayList<Integer> calculateStartingIndexes(String
ciphertext, String trigram) {
        ArrayList<Integer> indexes = new ArrayList<>();
        int index = -1;
        while ((index = ciphertext.indexOf(trigram, index + 1)) != -1) {
            indexes.add(index);
        }

        return indexes;
    }




 // Step 4: Find the length of the key using GCD of distances
 public static int findKeyLength(String ciphertext) {
    HashMap<String, Integer> trigramMap = findMostFrequentTrigram(ciphertext);

    String maxTrigram = findMaximumOccurringTrigram(trigramMap);

    ArrayList<Integer> startingIndexes = calculateStartingIndexes(ciphertext,
maxTrigram);
    ArrayList<Integer> distances = new ArrayList<>();

    for (int i = 1; i < startingIndexes.size(); i++) {
        int distance = startingIndexes.get(i) - startingIndexes.get(i - 1);
        distances.add(distance);
    }

    // Find the greatest common divisor (GCD) of distances
```

```java
    int keyLength = calculateGCD(distances);
    return keyLength;
}

// Helper function to calculate the GCD of a list of integers
public static int calculateGCD(ArrayList<Integer> numbers) {
    if (numbers.isEmpty()) {
        return 1; // Default to a key length of 1 if no distances are found
    }

    int gcd = numbers.get(0);
    for (int i = 1; i < numbers.size(); i++) {
        gcd = calculateGCD(gcd, numbers.get(i));
    }
    return gcd;
}

// Helper function to calculate the GCD of two integers using Euclid's
algorithm
private static int calculateGCD(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return Math.abs(a);
}

// Step 5: Make substrings and get a key for each substring using index of
coincidence
public static String findKey(String ciphertext, int keyLength) {
    // Implement this function
    // Split the ciphertext into keyLength substrings
    // Calculate the key for each substring using index of coincidence
    // Combine the keys to form the final key
    StringBuilder finalKey = new StringBuilder();

    for (int i = 0; i < keyLength; i++) {
        String substring = extractSubstring(ciphertext, i, keyLength);
        //System.out.println("substring :-  "+ substring);

        int substringKey = breakShiftCipher(substring);
        finalKey.append((char) ('a' + substringKey));
    }
    return finalKey.toString();
}
```

```java
// Helper function to extract a substring starting from index with a given
step
private static String extractSubstring(String text, int startIndex, int step)
{
    StringBuilder substring = new StringBuilder();
    for (int i = startIndex; i < text.length(); i += step) {
        substring.append(text.charAt(i));
    }
    return substring.toString();
}


///===----------------------------------------------------








///==----------------------------------------

public static int breakShiftCipher(String ciphertext) {
    // Calculate letter frequencies in the ciphertext
    Map<Character, Integer> letterFrequencies = new HashMap<>();
    int totalLetters = 0;

    for (char c : ciphertext.toCharArray()) {
        if (Character.isLetter(c)) {
            char uppercaseChar = Character.toUpperCase(c);
            letterFrequencies.put(uppercaseChar,
letterFrequencies.getOrDefault(uppercaseChar, 0) + 1);
            totalLetters++;
        }
    }

    // Calculate Ij values for all shifts (0 to 25)
    double[] ijValues = new double[26];

    for (int j = 0; j < 26; j++) {
        for (char pi : EnglishLetterFrequencies.letterFrequencies.keySet()) {
            char qi = (char) (((pi - 'a' + j) % 26) + 'a');

            double pij = (double) letterFrequencies.getOrDefault(qi, 0) /
totalLetters;
            ijValues[j] += EnglishLetterFrequencies.letterFrequencies.get(pi)
* pij;
        }
    }
```

```
    // Find the key with the closest Ij value to the expected value
    double targetIj = 0.065;
    int bestKey = -1;
    double closestIj = Double.MAX_VALUE;

    for (int key = 0; key < 26; key++) {
        double diff = Math.abs(ijValues[key] - targetIj);
        if (diff < closestIj) {
            closestIj = diff;
            bestKey = key;
        }
    }

    return bestKey;
}


   // Step 6: Decrypt the Vigenère cipher using the key
public static String decryptVigenere(String ciphertext, String key) {
    StringBuilder plaintext = new StringBuilder();
    int keyLength = key.length();

    for (int i = 0; i < ciphertext.length(); i++) {
        char ciphertextChar = ciphertext.charAt(i);
        char keyChar = key.charAt(i % keyLength);

        if (Character.isLetter(ciphertextChar)) {
            // Determine whether the character is uppercase or lowercase
            boolean isUpperCase = Character.isUpperCase(ciphertextChar);
            char base = isUpperCase ? 'A' : 'a';

            // Perform the Vigenère decryption
            char decryptedChar = (char) (((ciphertextChar - keyChar + 26) %
26) + base);
            plaintext.append(decryptedChar);
        } else {
            // If the character is not a letter, leave it unchanged
            plaintext.append(ciphertextChar);
        }
    }

    return plaintext.toString();
}

}
```

OUTPUT

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\aayus\Desktop\Sem-5\Applied Cryptography\Assignments>  & 'C:\Program Files\Java\jdk-19\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\
Users\aayus\AppData\Roaming\Code\User\workspaceStorage\7807b792ea56be5f5964aa7b62aba17f\redhat.java\jdt_ws\Assignments_61eefebc\bin' 'VigenereCipherCracker'
Key length is 4
key is  cool
Decrypted Text: greeneryinallitslushnessandvibrancyisanintegralpartofournaturalenvironmentitencompassesthemyriadoftreesshrubsplantsandgrassesthatblanketourlandscapes
providingmorethanjustaestheticappealitplaysapivotalroleinourlivesinfluencingourphysicalhealthmentalwellbeingandthesustainabilityofourplanetatitscoregreeneryisasymbol
oflifeandvitalityitisthelivingcanvasthatpaintstheworldaroundusfromthetoweringforeststhatwhispersecretsoftimelessnesstothedelicatepetalsofaspringblossomgreenerycaptur
estheessenceofgrowthrenewalandharmonywithnatureyetitssignificancegoesfarbeyondmereaestheticsoneofthemostevidentadvantagesofgreeneryisitspositiveimpactontheenvironmen
ttreesandplantsactasnaturalairpurifiersabsorbingharmfulpollutantssuchascarbondioxideandreleasingoxygentheymitigatetheeffectsofclimatechangebyreducinggreenhousegasemi
ssionsandstabilizingtemperaturesmoreovertherootsystemsofplantshelppreventsoilerosionensuringthelongtermfertilityofthelandgreeneryalsoplaysacrucialroleinbiodiversityd
iverseecosystemsthrivewherethereisanabundanceofplantlifeprovidinghabitatsforcountlessspeciesofanimalsandinsectsthisinterconnectedweboflifeensuresthebalanceofnaturean
dthesurvivalofmanyspeciesincludingourowninurbanareasgreenerytakesonaddedsignificanceparksandgreenspacesprovidearespitefromtheconcretejungleofferingpeopleaplacetorela
xexerciseandconnectwithnatureaccesstogreenareaswithincitieshasbeenlinkedtoimprovedmentalhealthreducedstresslevelsandincreasedoverallwellbeingasurbanizationcontinuest
orisepreservingandexpandinggreenspacesisessentialforthehealthandhappinessofcitydwellersfurthermoregreeneryhaseconomicbenefitstheagriculturalindustryreliesheavilyonth
ecultivationofcropsandthecareoflivestockwhichdependonfertilesoilandahospitableclimategreeneryhelpsmaintaintheseconditionsensuringastablefoodsupplyandsupportingliveli
hoodstheimportanceofgreenerycannotbeoverstatedwhenitcomestoeducationandawarenessitservesasalivingclassroomteachingusaboutthecyclesoflifeecologyandthedelicatebalanceo
fecosystemsgreenspacesofferopportunitiesforchildrentoconnectwithnaturefosteringasenseofresponsibilityandstewardshipfortheenvironmentfromanearlyagehoweverdespiteitsma
nyadvantagesgreeneryfacesthreatsintheformofdeforestationhabitatdestructionandpollutionasstewardsofthisplanetitisourresponsibilitytoprotectandpreservethegreenerythats
ustainsusinitiativessuchasafforestationreforestationandsustainablelandmanagementarecrucialstepstowardensuringthecontinuedvitalityofourgreenspacesinconclusiongreenery
isnotmerelyanembellishmentinthetapestryofourworlditisthethreadthatbindsustothenaturalworldandsustainsourexistencefromtheairwebreathetothementalsolaceitprovidesgreene
ryenrichesourlivesincountlesswaysaswemoveforwardletuscelebrateprotectandcultivategreeneryrecognizingitsprofoundimportancetoourhealthhappinessandthefutureofourplanet
PS C:\Users\aayus\Desktop\Sem-5\Applied Cryptography\Assignments>
```

Problem 2: The shift cipher can also be defined over the 128-character ASCII alphabet (rather than the 26-character English alphabet). a. Provide a formal definition of the scheme in this case. b. Discuss how the attack we have studied can be modified to break the modified scheme.

Solution:-

a. Formal Definition of the 128-Character ASCII Shift Cipher: The 128-character ASCII shift cipher is a symmetric encryption scheme that operates on a message consisting of characters from the 128-character ASCII character set. It uses a fixed integer value, often called the "shift" or "key," to perform a simple modular addition operation on each character in the message to produce the ciphertext.

9. Key Generation:
    - Choose an integer key (K) from the range [0, 127]. This key determines the amount by which each character in the plaintext will be shifted.
10. Encryption:
    - For each character (P) in the plaintext message:
        - Compute the ciphertext character (C) using the formula: C = (P + K) mod 128

11. Decryption:

- To decrypt the ciphertext and obtain the original plaintext:
  - For each character (C) in the ciphertext:
    - Compute the plaintext character (P) using the formula: $P = (C - K + 128) \bmod 128$

Example: Let's say we want to encrypt the message "HELLO" using a key (K) of 10 in the 128-character ASCII shift cipher. The ASCII values of the characters are:

- H: 72
- E: 69
- L: 76
- L: 76
- O: 79

Encryption:

- H (72) + 10 ≡ 82 (R)
- E (69) + 10 ≡ 79 (O)
- L (76) + 10 ≡ 86 (V)
- L (76) + 10 ≡ 86 (V)
- O (79) + 10 ≡ 89 (Y)

So, the encrypted message is "ROVVY."

b. Discussion of Attacks on the Modified 128-Character ASCII Shift Cipher: The modified 128-character ASCII shift cipher is susceptible to various attacks, including:

1. **Frequency Analysis:** In the ASCII shift cipher, certain characters may still occur more frequently, even though there are 128 possible characters. This allows attackers to use frequency analysis to make educated guesses about the key. The Index of Coincidence (IC) can be used to identify deviations from randomness in the ciphertext, and if the IC is notably different from 1/128, it may indicate the presence of a Caesar cipher. Attackers can then employ frequency analysis to determine the key.

Example: If the character ' ' (space) is the most frequent character in the ciphertext, an attacker may guess that it corresponds to 'e' in the plaintext and calculate the shift value accordingly.

2. **Kasiski Test:** The Kasiski Test can be adapted to the 128-character ASCII shift cipher. Attackers can look for repeated sequences of characters in the ciphertext, calculate the distances between them, and search for common factors in these distances. This can help them guess the length of the key (shift value) and apply frequency analysis to break the cipher.

Example: If the ciphertext contains the repeated sequence "XYZ" with distances of 10 and 20 characters, the common factor is 10, suggesting a key with a shift value of 10.

Despite the larger character set, the 128-character ASCII shift cipher remains vulnerable to these classic cryptographic attacks, making it relatively insecure for secure communication.