Ministry of Education, Culture and Research of the Republic of Moldova
Technical University of Moldova
Department of Software and Automation Engineering

REPORT

Laboratory work No. 3

Discipline: Cryptography and Security

Elaborated: FAF-221,
Revenco Victor

Checked: asist. univ.,
Dumitru Nirca

Chișinău 2024

Topic: Polialphabetic Cipher

Tasks:

De implementat algoritmul Vigenere în unul din limbajele de programare pentru mesaje în limba română (31 de litere), acestea fiind codificate cu numerele 0, 1, ... 30. Valorile caracterelor textului sunt cuprinse între ‘A’ și ’Z’, ’a’ și ’z’ și nu sunt premise alte valori. În cazul în care utilizatorul introduce alte valori - i se va sugera diapazonul corect al caracterelor. Lungimea cheii nu trebuie să fie mai mică de 7. Criptarea și decriptarea se va realiza în conformitate cu formulele din modelul matematic prezentat mai sus. În mesaj mai întâi trebuie eliminate spațiile, apoi toate literele se vor transforma în majuscule. Utilizatorul va putea alege operația - criptare sau decriptare, va putea introduce cheia, mesajul sau criptograma și va obține criptograma sau mesajul decriptat.

Theoretical notes:

Cifruri de substituție polialfabetică (polyalphabetic ciphers). Slăbiciunea cifrurilor monoalfabetice este definită de faptul că distribuția lor de frecvență reflectă distribuția alfabetului folosit. Un cifru este mai sigur din punct de vedere criptografic dacă prezintă o distribuție cât mai regulată, care să nu ofere informații criptanalistului. O cale de a aplatiza distribuția este combinarea distribuțiilor ridicate cu cele scăzute. Daca litera T este criptată câteodată ca a și altă dată ca b, și dacă litera X este de asemenea câteodată criptată ca a și altă dată ca b, frecvența ridicată a lui T se combină cu frecvența scăzută a lui X producând o distribuție mai moderata pentru a și pentru b. Două distribuții se pot combina prin folosirea a doua alfabete separate de criptare, primul pentru caracterele aflate pe poziții pare în textul clar, al doilea pentru caracterele aflate pe poziții impare, rezultând necesitatea de a folosi alternativ doua tabele de translatare, de exemplu permutările

$p1(a)=(3\cdot a) \bmod 26$ și $p2(a)=(7\cdot a +13) \bmod 26$. Â

Diferența dintre cifrurile polialfabetice și cele monoalfabetice constă în faptul că substituția unui caracter variază în text, în funcție de diverși parametri (poziție, context etc.). Aceasta conduce bineînțeles la un număr mult mai mare de chei posibile. Se consideră că primul sistem de criptare polialfabetic a fost creat de Leon Battista în 1568. Unele aplicații actuale folosesc încă pentru anumite secțiuni astfel de sisteme de criptare.

Metoda de criptare cunoscută sub numele de „cifrul Vigenère” a fost atribuită greșit lui Blaise de Vigenère în secolul al XIX-lea și, de fapt, a fost descrisă pentru prima dată de Giovan Battista Bellaso în cartea sa din 1553 La cifra del. Sig. Vigenère a creat un cifru asemănător, dar totuși diferit și mai puternic în 1586. Pe da altă parte, cifrul Vigenere folosește aceleași operații ca și cifrul Cezar. Cifrul Vigenere și fel deplasează literele, dar, spre deosebire de Cezar, nu se poate sparge ușor în 26 combinații. Cifrul Vigenere folosește o deplasare multiplă. Cheia nu este constituită de o singură deplasare, ci de mai multe, fiind generate de câțiva întregi ki, unde $0 \leq ki \leq 25$, dacă luăm ca reper alfabetul latin cu 26 de litere. Criptarea se face în felul următor: ci = (mi + ki) mod 26. Cheia poate fi, de exemplu, k = (5, 20, 17, 10, 20, 13) și ar provoca deplasarea primei litere cu 5, c1=m1 + 5 (mod 26), a celei de a doua cu 20, c2 = m2 + 20 (mod 26), ș.a.m.d. până la sfârșitul cheii și apoi de la început, din nou. Cheia este de obicei un cuvânt, pentru a fi mai ușor de memorat – cheia de mai sus corespunde cuvântului „furtun”. Metoda cu deplasare multiplă oferă protecție suplimentară din două motive:

• primul motiv este că ceilalți nu cunosc lungimea cheii;

• cel de al doilea motiv este că numărul de soluții posibile crește odată cu mărimea cheii; de exemplu, pentru lungimea cheii egală cu 5, numărul de combinații care ar fi necesare la căutarea exhaustivă ar fi 26^5 = 11 881 376.

Decriptarea pentru cifrul Vigenere este asemănătoare criptării. Diferența constă în faptul că se scade cheia din textul cifrat,

$$mi = (ci - ki) \bmod 26.$$

Implementation (Vigenere Cipher)

```
public static String normalizeText(String text) {
    StringBuilder normalized = new StringBuilder();
    for (char c : text.toUpperCase().toCharArray()) {
        if (ROMANIAN_ALPHABET.indexOf(c) != -1) {
            normalized.append(c);
        }
    }
    return normalized.toString();
}
```

First of all we add a method called ***normalizeText*** which converts the text to upper case and removes any characters not found in the Romanian alphabet.

```
public static String encrypt(String text, String key) {
    text = normalizeText(text);
    key = normalizeText(key);
    StringBuilder result = new StringBuilder();

    for (int i = 0, j = 0; i < text.length(); i++) {
        char c = text.charAt(i);
        int shift = ROMANIAN_ALPHABET.indexOf(key.charAt(j %
key.length()));
        int originalPos = ROMANIAN_ALPHABET.indexOf(c);
        int newPos = (originalPos + shift) % ROMANIAN_ALPHA-
BET.length();
        result.append(ROMANIAN_ALPHABET.charAt(newPos));
        j++;
    }

    return result.toString();
}
```

In the ***encrypt*** method we give both the text and the key. Both of these go through the ***normalizeText*** method. We then initialize a result variable. We go through each char in the text, and shift to the corresponding char in the key. Then the result is the char at the new position.

```
public static String decrypt(String text, String key) {
    text = normalizeText(text);
    key = normalizeText(key);
    StringBuilder result = new StringBuilder();

    for (int i = 0, j = 0; i < text.length(); i++) {
        char c = text.charAt(i);
        int shift = ROMANIAN_ALPHABET.indexOf(key.charAt(j %
key.length()));
        int originalPos = ROMANIAN_ALPHABET.indexOf(c);
        int newPos = (originalPos - shift + ROMANIAN_ALPHA-
BET.length()) % ROMANIAN_ALPHABET.length();
        result.append(ROMANIAN_ALPHABET.charAt(newPos));
        j++;
    }

    return result.toString();
}
```

In the ***decrypt*** method we also give the text and the key which both go through the ***normalizeText*** method. Then again we initialize a result. We get the shift and original position, after which we find the new position, and append that char to the result.

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Do you want to Encrypt(1) or De-
crypt(2)? ");
    String choice = scanner.nextLine().trim().toUpperCase();

    System.out.print("Enter the text: ");
    String text = scanner.nextLine();

    System.out.print("Enter the key (minimum 7 characters):
");
    String key = scanner.nextLine();

    if (key.length() < 7) {
        System.out.println("Key must be at least 7 characters
long.");
        return;
    }

    if (choice.equals("1")) {
        String encryptedText = encrypt(text, key);
        System.out.println("Encrypted text: " + encrypt-
edText);
    } else {
        String decryptedText = decrypt(text, key);
        System.out.println("Decrypted text: " + decrypt-
edText);
    }

    scanner.close();
}
```

In the ***main*** method we give the user the choice to encrypt/decrypt after which we ask for the text and key (which should be min 7 char length). Depending on his choice we call the corresponding method.