Учреждение образования

«Белорусский государственный технологический университет»

**Кафедра информационных систем и технологий**

**«Отчёт по лабораторной работе №6»**

«Изучение устройства и функциональных особенностей шифровальной машины «Энигма»

**Выполнил:** студент 3 курса

4 группы специальности ПОИТ

Супрунюк Евгений Андреевич

**Проверил:** преподаватель

Сазонова Дарья Владимировна

Минск 2023

1. **Описание приложения**

Приложение-симулятор шифровальной машины, состоящей из клавиатуры, трех роторов и отражателя. Написано на языке программирования Python. Типы роторов (L-M-R) и отражателя Re следует выбрать из таблиц на рис. 1 в соответствии со своим вариантом, представленным в таблице 1. Крайний правый столбец этой таблицы показывает, на какое число шагов (букв, i) перемещается соответствующий ротор при зашифровании одного (текущего) символа; число 0 означает перемещение соответствующего ротора на один шаг при условии, что расположенный правее ротор совершит один оборот.

Таблица 1.1 – Данные варианта.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вариант задания | L | M | R | Re | Li-Mi-Ri |
| 13 | IV | III | II | C Dunn | 0-0-4 |

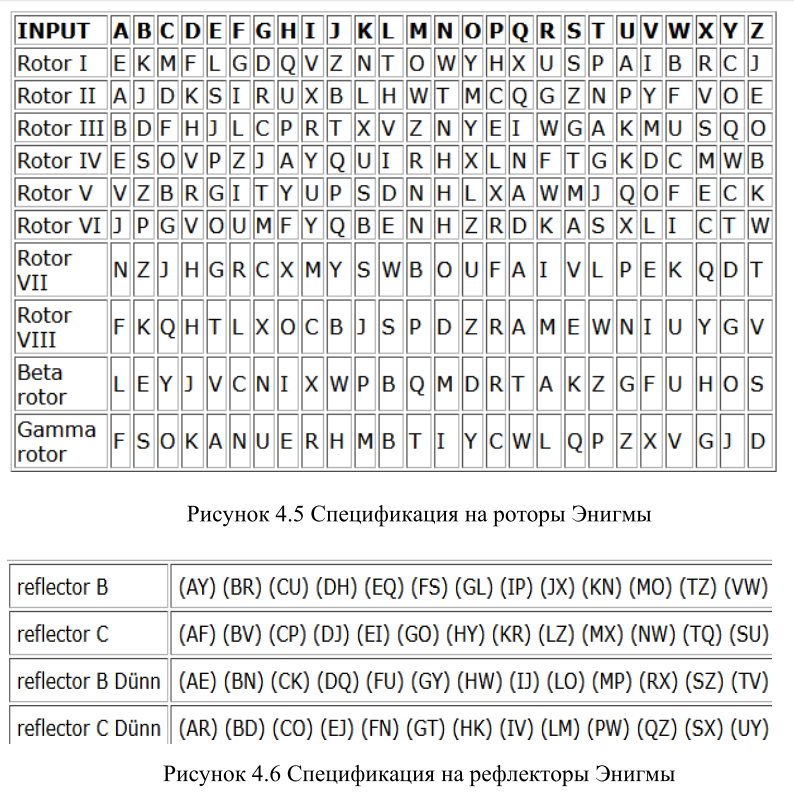


Рисунок 1.1 – спецификация для выполнения задания

1. **Методика выполнения расчетов**

В данной лабораторной работе была поставлена цель создания приложение-симулятора шифровальной машины, состоящей из клавиатуры, трех роторов и отражателя. На листинге 2.1 представлены функции, реализующие данную функциональность.

|  |
| --- |
| def get\_rotors(list\_rotors):      dict\_rotors = {'I': 'EKMFLGDQVZNTOWYHXUSPAIBRCJ',                     'II': 'AJDKSIRUXBLHWTMCQGZNPYFVOE',                     'III': 'BDFHJLCPRTXVZNYEIWGAKMUSQO',                     'IV': 'ESOVPZJAYQUIRHXLNFTGKDCMWB',                     'V': 'VZBRGITYUPSDNHLXAWMJQOFECK',                     'VI': 'JPGVOUMFYQBENHZRDKASXLICTW',                     'VII': 'NZJHGRCXMYSWBOUFAIVLPEKQDT',                     'VIII': 'FKQHTLXOCBJSPDZRAMEWNIUYGV',                     'Beta': 'LEYJVCNIXWPBQMDRTAKZGFUHOS',                     'Gamma': 'FSOKANUERHMBTIYCWLQPZXVGJD'}      return {f'{key}': dict\_rotors[key] for key in dict\_rotors.keys() if key in list\_rotors}  def get\_reflector(reflector):      dict\_reflectors = {'B': ['AY', 'BR', 'CU', 'DH', 'EQ', 'FS', 'GL', 'IP', 'JX', 'KN', 'MO', 'TZ', 'VW'],                         'C': ['AF', 'BV', 'CP', 'DJ', 'EI', 'GO', 'HY', 'KR', 'LZ', 'MX', 'NW', 'TQ', 'SU'],                         'B Dunn': ['AE', 'BN', 'CK', 'DQ', 'FU', 'GY', 'HW', 'IJ', 'LO', 'MP', 'RX', 'SZ', 'TV'],                         'C Dunn': ['AR', 'BD', 'CO', 'EJ', 'FN', 'GT', 'HK', 'IV', 'LM', 'PW', 'QZ', 'SX', 'UY']}      return {f'{key}': dict\_reflectors[key] for key in dict\_reflectors.keys() if key == reflector}  def replace\_non\_occurring\_characters\_and\_uppercase(message, replacement\_symbol):      non\_occurring\_symbol = []      if not replacement\_symbol.isalpha():          replacement\_symbol = 'X'      for symbol in message:          if symbol not in non\_occurring\_symbol and not symbol.isalpha():              non\_occurring\_symbol.append(symbol)      for i in non\_occurring\_symbol:          message = message.replace(i, replacement\_symbol)      return message.upper()  def set\_rotors\_initial\_position(rotors, initial\_position):      alphabet = string.ascii\_uppercase      for index, rotor in enumerate(rotors.keys()):          rotors[rotor] = rotors[rotor][alphabet.index(initial\_position[index]):] +\                          rotors[rotor][0:alphabet.index(initial\_position[index])]      return rotors  def get\_correct\_rotors\_steps(steps\_rotors):      correct\_steps\_rotors = []      for i in range(0, len(steps\_rotors) - 1):          if steps\_rotors[i] == 0 and steps\_rotors[i + 1] != 0:              correct\_steps\_rotors.append(1)          else:              correct\_steps\_rotors.append(steps\_rotors[i])      correct\_steps\_rotors.append(steps\_rotors[len(steps\_rotors) - 1])      return correct\_steps\_rotors  def encrypt\_enigma(message, list\_rotors, reflector, replacement\_symbol='X',                     steps\_rotors=(1, 1, 1), initial\_position='AAA'):      rotors\_direct = set\_rotors\_initial\_position(get\_rotors(list\_rotors), initial\_position)      rotors\_reverse = {k: v for k, v in reversed(list(rotors\_direct.items()))}      rotors\_direct\_output = set\_rotors\_initial\_position(get\_rotors(list\_rotors), initial\_position)      rotors\_reverse\_output = {k: v for k, v in reversed(list(rotors\_direct.items()))}      reflector = get\_reflector(reflector)      message = replace\_non\_occurring\_characters\_and\_uppercase(message, replacement\_symbol)      steps\_rotors = get\_correct\_rotors\_steps(steps\_rotors)      alphabet = string.ascii\_uppercase      encrypt\_message = ''      for symbol in message:          for key in rotors\_reverse.keys():              symbol = rotors\_reverse[key][alphabet.index(symbol)]          for key in reflector.keys():              if ''.join(reflector[key]).index(symbol) % 2 == 1:                  symbol = reflector[key][''.join(reflector[key]).index(symbol) // 2][0]              else:                  symbol = reflector[key][''.join(reflector[key]).index(symbol) // 2][1]          for key in rotors\_direct.keys():              symbol = alphabet[rotors\_direct[key].index(symbol)]          encrypt\_message += symbol          for index, key in enumerate(rotors\_direct.keys()):              rotors\_direct[key] = rotors\_direct[key][steps\_rotors[index]:] + rotors\_direct[key][0: steps\_rotors[index]]          rotors\_reverse = {k: v for k, v in reversed(list(rotors\_direct.items()))}      print(f'Сообщение для зашифровки: {message}\n'            f'Заменяющий символ: {replacement\_symbol}\n'            f'Роторы (прямой порядок): {rotors\_direct\_output}\n'            f'Роторы (обратный порядок): {rotors\_reverse\_output}\n'            f'Рефлектор: {reflector}\n'            f'Шаги роторов: {steps\_rotors}\n'            f'Начальная позиция роторов: {initial\_position}\n'            f'Зашифрованное сообщение: {encrypt\_message}\n')      return encrypt\_message  def decrypt\_enigma(encrypt\_message, list\_rotors, reflector, steps\_rotors, initial\_position):      rotors\_direct = set\_rotors\_initial\_position(get\_rotors(list\_rotors), initial\_position)      rotors\_reverse = {k: v for k, v in reversed(list(rotors\_direct.items()))}      rotors\_direct\_output = set\_rotors\_initial\_position(get\_rotors(list\_rotors), initial\_position)      rotors\_reverse\_output = {k: v for k, v in reversed(list(rotors\_direct.items()))}      reflector = get\_reflector(reflector)      steps\_rotors = get\_correct\_rotors\_steps(steps\_rotors)      alphabet = string.ascii\_uppercase      decrypt\_message = ''      for symbol in encrypt\_message:          for key in rotors\_reverse.keys():              symbol = rotors\_reverse[key][alphabet.index(symbol)]          for key in reflector.keys():              if ''.join(reflector[key]).index(symbol) % 2 == 1:                  symbol = reflector[key][''.join(reflector[key]).index(symbol) // 2][0]              else:                  symbol = reflector[key][''.join(reflector[key]).index(symbol) // 2][1]          for key in rotors\_direct.keys():              symbol = alphabet[rotors\_direct[key].index(symbol)]          decrypt\_message += symbol          for index, key in enumerate(rotors\_direct.keys()):              rotors\_direct[key] = rotors\_direct[key][steps\_rotors[index]:] + rotors\_direct[key][0: steps\_rotors[index]]              rotors\_reverse = {k: v for k, v in reversed(list(rotors\_direct.items()))}      print(f'Сообщение для расшифровки: {encrypt\_message}\n'            f'Роторы (прямой порядок): {rotors\_direct\_output}\n'            f'Роторы (обратный порядок): {rotors\_reverse\_output}\n'            f'Рефлектор: {reflector}\n'            f'Шаги роторов: {steps\_rotors}\n'            f'Начальная позиция роторов: {initial\_position}\n'            f'Расшифрованное сообщение: {decrypt\_message}')      return decrypt\_message |

Листинг 2.1 –код программы, реализующая приложение-симулятора шифровальной машины «Энигмы»

**3. Результаты работы приложения**

Для выполнения расчетов достаточно необходимо запустить приложение. Рисунок 3.1 показывают требуемые в данной лабораторной работе результаты. Внизу также предствалены расчет криптостойкости машины.

5\*4\*3=60 (количество возможных комбинаций роторов)

26\*26\*26=17576 (возможные положения роторов)

26\*26=676 (начальные положения роторов)

26! / (26-2\*10) \*10! \* 210=1,5 \*1014 (коммутация панели)

60\*17576\*676\*1,5\*1014 =1,07\*1023

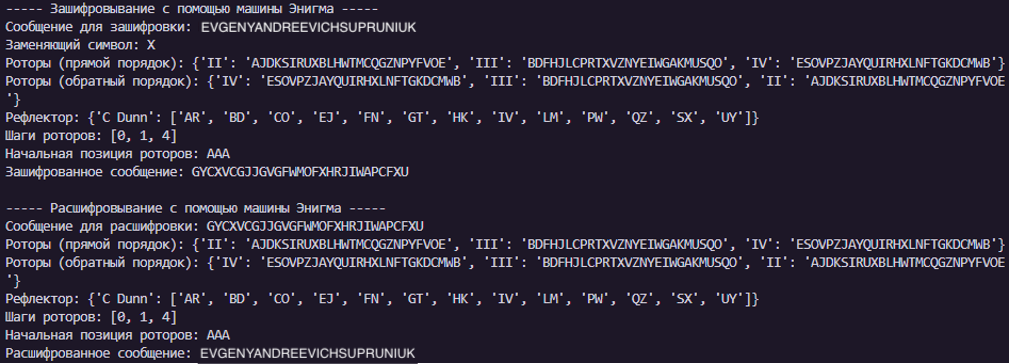


Рисунок 3.1 – Результат работы по приложение-симулятора шифровальной машины Энигмы

**Вывод**

В ходе лабораторной работы была разработана реализация шифровальной машины Энигма на языке Python. Также установлено, что криптостойкость машины составляет 1,07\*1023 комбинаций.