

Министерство науки и высшего образования РФ
Федеральное государственное автономное образовательное
учреждение
высшего образования
«Московский политехнический университет»
(Московский политех)

Отчёт по курсу «Программирование криптографических алгоритмов»
Лабораторная работа 1.5. Шифры граммирования



Выполнил:

Студент группы 221-352

Иванов В. В.

Проверил преподаватель: Бутакова Н. Г.

Москва 2024г.

Аннотация

- **Среда программирования**

- Visual Studio Code

- **Язык программирования**

- Python

- **Процедуры для запуска программы**

- Visual Studio Code (main.py)

- **Пословица-тест**

- Тот, кто ложится на два стула, падает на ребра.

- **Текст для проверки работы (не меньше 1000 знаков (1430))**

Жизнь - это удивительное приключение, полное разнообразных событий и встреч. В каждом моменте мы находим что-то новое и уникальное. Стремление к росту и саморазвитию вдохновляет нас на поиск новых горизонтов. Важно помнить, что каждый шаг вперед приносит с собой уроки и опыт.

Разнообразие культур, языков и традиций делает наш мир удивительно богатым. Общение с людьми разных национальностей расширяет кругозор, позволяя нам понимать и уважать друг друга. Взаимное уважение и терпимость создают основу для гармоничного сосуществования.

Природа тоже играет важную роль в нашей жизни. Красота закатов, шум океана, пение птиц - все это напоминает нам о величии мира природы. Забота о окружающей среде становится неотъемлемой частью ответственного образа жизни.

Работа и творчество придают смысл нашим усилиям. Стремление к достижению целей мотивирует нас на новые начинания. Каждый проект, даже самый маленький, приносит удовлетворение и чувство выполненного долга.

Семья и друзья являются надежной опорой в нашей жизни. Обмен историями, веселые посиделки и поддержка в трудные моменты создают теплую атмосферу взаимопонимания и любви.

Таким образом, наша жизнь - это мозаика различных моментов, соединенных воедино. Важно ценить каждый момент и стремиться делать мир вокруг нас ярче и лучше. С любовью, терпением и целеустремленностью мы можем создавать свою уникальную историю, наполненную смыслом и радостью.

• Код программы-интерфейса

```
• import sys
• from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QPushButton, QComboBox,
  QTextEdit, QCheckBox
• from PyQt5.QtCore import Qt
• from atbash import atbash_encrypt, atbash_decrypt
• from cesar import cesar_encrypt, cesar_decrypt, cesar_check_parameters
• from polibia import polibia_encrypt, polibia_decrypt
• from tritemiy import tritemiy_encrypt, tritemiy_decrypt
• from belazo import belazo_encrypt, belazo_decrypt, belazo_check_parameters
• from vigenere import vigenere_encrypt, vigenere_decrypt, vigenere_check_parameters
• from S_block import s_block_check_parameters, s_block_encrypt, s_block_decrypt
• from matrix import matrix_encrypt, matrix_decrypt, matrix_check_parameters
• from playfair import playfair_encrypt, playfair_decrypt, playfair_check_parameters
• from veritcalTransposition import vertical_transposition_check_parameters, vertical_transposition_encrypt,
  vertical_transposition_decrypt
• from cardanosGrid import cardanosGridCheckParameters, cardanosGridEncrypt, cardanosGridDecrypt
• from feistelsNetwork import feistelsNetworkCheckParameters, feistelsNetwork
• from shannons_notebook import shannonsNotebookCheckParameters, shannonsNotebookEncrypt, shannonsNotebookDecrypt
• from hexoize import StrToHex, HexToStr
• from magma import MagmaCheckParameters, magma
•
• available_ciphers = [
•     "Шифр АТБАШ", "Шифр Цезаря", "Шифр Полибия",
•     "Шифр Тритемия", "Шифр Белазо", "Шифр Виженера", "МАГМА(s_block)",
•     "Шифр Матричный", "Шифр Плейфера", "Шифр вертикальной перестановки",
•     "Шифр решетка Кардано", "Шифр сеть Фейстель", "Одноразовый блокнот Шеннона",
•     "Нехоize", "Магма (гаммирование)",
• ]
•
• alphabet = [
•     "а", "б", "в", "г", "д", "е", "ж", "з", "и", "й", "к", "л", "м",
```

```

•     "н", "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ",
•     "ъ", "ы", "ь", "э", "ю", "я"
• ]
•
• alphabet_polibia = [
•     ["а", "б", "в", "г", "д", "е"],
•     ["ж", "з", "и", "й", "к", "л"],
•     ["м", "н", "о", "п", "р", "с"],
•     ["т", "у", "ф", "х", "ц", "ч"],
•     ["ш", "щ", "ъ", "ы", "ь", "э"],
•     ["ю", "я"]
• ]
•
• alphabet_playfair = [
•     "а", "б", "в", "г", "д", "е", "ж", "з", "и", "к", "л", "м", "н",
•     "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ", "ъ",
•     "ы", "э", "ю", "я"
• ]
•
• alphabet_sblock = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "a", "b", "c", "d", "e", "f"]
•
• mem = {
•     "bigTextFlag": False,
•     "vigenereSwitch": False,
•     "mode": "encrypt",
• }
•
• class CipherApp(QWidget):
•     def __init__(self):
•         super().__init__()
•         self.initUI()

```

```

def initUI(self):
    self.setWindowTitle('Шифры')
    self.resize(960, 640)
    layout = QVBoxLayout()

    # Выбор шифра
    cipher_layout = QHBoxLayout()
    cipher_label = QLabel('Выберите шифр:')
    self.cipher_combo = QComboBox()
    self.cipher_combo.addItem('Введите открытый текст(Расшифрованный):')
    cipher_layout.addWidget(cipher_label)
    cipher_layout.addWidget(self.cipher_combo)

    # Ввод открытого текста
    open_text_label = QLabel('Введите открытый текст(Расшифрованный):')
    self.open_text_edit = QTextEdit()

    # Ввод зашифрованного текста
    cipher_text_label = QLabel('Шифрованный текст:')
    self.cipher_text_edit = QTextEdit()

    # Ввод сдвига для шифра Цезаря
    self.cesar_shift_edit = QLineEdit()
    self.cesar_shift_edit.setPlaceholderText('Введите сдвиг для шифра Цезаря')
    self.cesar_shift_edit.textChanged.connect(self.check_cesar_shift)

    # Ввод ключевого слова для шифра Белазо или Плейфера
    self.keyword_edit = QLineEdit()
    self.keyword_edit.setPlaceholderText('Введите ключевое слово для шифра Белазо, Плейфера, Вертикальной перестановки')

    # Ввод ключевой буквы для шифра Виженера

```

```

• self.vigener_key_edit = QLineEdit()
• self.vigener_key_edit.setPlaceholderText('Введите ключевую букву для шифра Виженера')
• self.vigener_key_edit.textChanged.connect(self.check_vigener_key)
•
• # Ввод ключевой матрицы для шифра Матричный
• self.matrix_edit = QLineEdit()
• self.matrix_edit.setPlaceholderText('Введите ключевую матрицу для шифра Матричный')
•
• # Ввод размерности решётки для шифра Кардано
• self.cardanosGrid_edit = QLineEdit()
• self.cardanosGrid_edit.setPlaceholderText('Введите размерность решётки для шифра Кардано (8 8)')
•
• # Ввод решётки для шифра Кардано
• self.cardanosGridGRIDGRID_edit = QLineEdit()
• self.cardanosGridGRIDGRID_edit.setPlaceholderText('Решётка (0 1, 1 6, 2 1, 2 2, 2 4, 3 6, 3 7, 4 3, 4 5, 5 0, 6
3, 6 5, 6 7, 7 2, 7 4, 7 7)')
•
• # Ввод решётки для шифра сеть Фейстеля
• self.feistelsNet_edit = QLineEdit()
• self.feistelsNet_edit.setPlaceholderText('Ключ для шифра сети Фейстеля')
•
• # Ввод t для Одноразового блокнота Шеннона
• self.shannons_t_a_c_edit = QLineEdit()
• self.shannons_t_a_c_edit.setPlaceholderText('t а с для Одноразового блокнота Шеннона')
•
• # Ввод ключа для Магмы (гаммирования)
• self.magmaKeyInput_edit = QLineEdit()
• self.magmaKeyInput_edit.setPlaceholderText('Ключ для Магма(гаммирования)')
•
• # Ввод Инициализирующий вектор
• self.magmaVectorInput_edit = QLineEdit()
• self.magmaVectorInput_edit.setPlaceholderText('Инициализирующий вектор для Магма(гаммирования)')

```

```

•
•     # Режим работы шифра (шифрование или дешифрование)
•     mode_layout = QHBoxLayout()
•     mode_label = QLabel('Выберите режим:')
•     self.mode_combo = QComboBox()
•     self.mode_combo.addItem('Шифрование')
•     self.mode_combo.addItem('Расшифрование')
•     mode_layout.addWidget(mode_label)
•     mode_layout.addWidget(self.mode_combo)
•
•
•     # Кнопка для запуска шифрования/дешифрования
•     self.encrypt_button = QPushButton('Выполнить')
•
•
•     layout.addLayout(cipher_layout)
•     layout.addWidget(open_text_label)
•     layout.addWidget(self.open_text_edit)
•     layout.addWidget(cipher_text_label)
•     layout.addWidget(self.cipher_text_edit)
•     layout.addWidget(self.cesar_shift_edit)
•     layout.addWidget(self.keyword_edit)
•     layout.addWidget(self.vigener_key_edit)
•     layout.addWidget(self.matrix_edit)
•     layout.addWidget(self.cardanosGrid_edit)
•     layout.addWidget(self.cardanosGridGRIDGRID_edit)
•     layout.addWidget(self.feistelsNet_edit)
•     layout.addWidget(self.shannons_t_a_c_edit)
•     layout.addWidget(self.magmaKeyInput_edit)
•     layout.addWidget(self.magmaVectorInput_edit)
•     layout.addLayout(mode_layout)
•     layout.addWidget(self.encrypt_button)
•
•
•     self.setLayout(layout)
•

```

```

•     # Переключатель для выбора режима текста
•     self.text_mode_checkbox = QCheckBox('Расширенный текст')
•     layout.addWidget(self.text_mode_checkbox)
•
•     # Подключение слотов к сигналам
•     self.encrypt_button.clicked.connect(self.cipher_parser)
•     self.text_mode_checkbox.stateChanged.connect(self.handle_text_mode_change)
•
•
•     def handle_text_mode_change(self, state):
•         if state == Qt.Checked:
•             mem["bigTextFlag"] = True
•         else:
•             mem["bigTextFlag"] = False
•
•     def check_cesar_shift(self):
•         shift_text = self.cesar_shift_edit.text()
•         try:
•             shift = int(shift_text)
•             if shift < 0 or shift >= len(alphabet):
•                 self.cesar_shift_edit.setStyleSheet("QLineEdit { color: red; }")
•             else:
•                 self.cesar_shift_edit.setStyleSheet("")
•         except ValueError:
•             self.cesar_shift_edit.setStyleSheet("QLineEdit { color: red; }")
•
•     def check_vigener_key(self):
•         key_text = self.vigener_key_edit.text()
•         if len(key_text) != 1 or key_text.lower() not in alphabet:
•             self.vigener_key_edit.setStyleSheet("QLineEdit { color: red; }")
•         else:
•             self.vigener_key_edit.setStyleSheet("")
•

```



```

def text_preparation(self, text):
    bigTextFlag = mem["bigTextFlag"]
    if bigTextFlag:
        # Обработка расширенного текста
        return text.replace("ё", "е").replace(".", "тчк").replace(",", "зпт").replace("-", "тире").replace(" ",
"прбл").replace(":", "двтч").replace(";", "тчсзн").replace("(", "отскб").replace(")", "зксб").replace("?",
"впрзн").replace("!", "восклзн").replace("\n", "првст").lower()
    else:
        # Обработка обычного текста
        return text.replace("ё", "е").replace(".", "тчк").replace(",", "зпт").replace("-", "тире").replace(" ",
""").replace(":", """).replace(";", """).replace("(", """).replace(")", """).replace("?", """).replace("!", """).replace("\n",
""").lower()

def cipher_parser(self):
    cipher_choose_input = self.cipher_combo.currentText()
    open_text_input = self.open_text_edit.toPlainText()
    cipher_text_input = self.cipher_text_edit.toPlainText()
    cesar_shift = self.cesar_shift_edit.text()
    keyword = self.keyword_edit.text()
    vigenere_keyletter = self.vigenere_key_edit.text()
    matrix_input = self.matrix_edit.text()
    cardanosGridSizeInput = self.cardanosGrid_edit.text()
    cardanosGridGRIDInput = self.cardanosGridGRIDGRID_edit.text()
    feistelsNetworkInput = self.feistelsNet_edit.text()
    shannonsNotebookT_A_C = self.shannons_t_a_c_edit.text()
    magmaKeyInput = self.magmaKeyInput_edit.text()
    magmaVectorInput = self.magmaVectorInput_edit.text()

    # Определение режима работы (шифрование или дешифрование)
    mode = 'encrypt' if self.mode_combo.currentText() == 'Шифрование' else 'decrypt'

    # Определение флага для обработки больших текстов

```

```

• bigTextFlag = len(open_text_input) > 1000 #ваш порог длины текста
•
• if cipher_choose_input == "Шифр АТБАШ":
•     if mode == "encrypt":
•         cipher_text_input = atbash_encrypt(self.text_preparation(open_text_input), alphabet)
•     elif mode == "decrypt":
•         open_text_input = atbash_decrypt(cipher_text_input, alphabet)
• elif cipher_choose_input == "Шифр Цезаря":
•     if cesar_shift: # Проверка на пустую строку
•         cesar_shift = int(cesar_shift)
•         if cesar_check_parameters(cesar_shift, alphabet):
•             if mode == "encrypt":
•                 cipher_text_input = cesar_encrypt(self.text_preparation(open_text_input), cesar_shift,
alphabet)
•             elif mode == "decrypt":
•                 open_text_input = cesar_decrypt(cipher_text_input, cesar_shift, alphabet)
•         else:
•             if mode == "encrypt":
•                 cipher_text_input = "Проверьте правильность ввода сдвига"
•             elif mode == "decrypt":
•                 open_text_input = "Проверьте правильность ввода сдвига"
•         else:
•             if mode == "encrypt":
•                 cipher_text_input = "Введите сдвиг для шифра Цезаря"
•             elif mode == "decrypt":
•                 open_text_input = "Введите сдвиг для шифра Цезаря"
• elif cipher_choose_input == "Шифр Полибия":
•     if mode == "encrypt":
•         cipher_text_input = polibia_encrypt(self.text_preparation(open_text_input), alphabet_polibia)
•     elif mode == "decrypt":
•         open_text_input = polibia_decrypt(cipher_text_input, alphabet_polibia)
• elif cipher_choose_input == "Шифр Тритемия":

```

```

•         if mode == "encrypt":
•             cipher_text_input = tritemiy_encrypt(self.text_preparation(open_text_input), alphabet)
•         elif mode == "decrypt":
•             open_text_input = tritemiy_decrypt(cipher_text_input, alphabet)
•     elif cipher_choose_input == "Шифр Белазо":
•         if keyword:
•             if belazo_check_parameters(keyword.lower(), alphabet):
•                 if mode == "encrypt":
•                     cipher_text_input = belazo_encrypt(self.text_preparation(open_text_input), keyword.lower(),
alphabet)
•                 elif mode == "decrypt":
•                     open_text_input = belazo_decrypt(cipher_text_input, keyword.lower(), alphabet)
•             else:
•                 if mode == "encrypt":
•                     cipher_text_input = "Проверьте правильность ввода ключевого слова"
•                 elif mode == "decrypt":
•                     open_text_input = "Проверьте правильность ввода ключевого слова"
•             else:
•                 if mode == "encrypt":
•                     cipher_text_input = "Введите ключевое слово для шифра Белазо"
•                 elif mode == "decrypt":
•                     open_text_input = "Введите ключевое слово для шифра Белазо"
•     elif cipher_choose_input == "Шифр Виженера":
•         if vigenere_keyletter:
•             if vigenere_check_parameters(vigenere_keyletter, alphabet):
•                 mode = "encrypt" if self.mode_combo.currentText() == 'Шифрование' else 'decrypt'
•                 if mode == "encrypt":
•                     cipher_text_input = vigenere_encrypt(self.text_preparation(open_text_input), vigenere_keyletter,
"selfkey", alphabet)
•                 elif mode == "decrypt":
•                     open_text_input = vigenere_decrypt(cipher_text_input, vigenere_keyletter, "selfkey", alphabet)
•             else:

```

```

•         if mode == "encrypt":
•             cipher_text_input = "Проверьте правильность ввода ключевой буквы"
•         elif mode == "decrypt":
•             open_text_input = "Проверьте правильность ввода ключевой буквы"
•
•     else:
•         if mode == "encrypt":
•             cipher_text_input = "Введите ключевую букву для шифра Виженера"
•         elif mode == "decrypt":
•             open_text_input = "Введите ключевую букву для шифра Виженера"
•     elif cipher_choose_input == "МАГМА(s_block)":
•         if mode == "encrypt":
•             if s_block_check_parameters(open_text_input, alphabet_sblock):
•                 cipher_text_input = s_block_encrypt(open_text_input, alphabet_sblock)
•             else:
•                 cipher_text_input = "Проверьте правильность ввода ключей"
•         elif mode == "decrypt":
•             if s_block_check_parameters(cipher_text_input, alphabet_sblock):
•                 open_text_input = s_block_decrypt(cipher_text_input, alphabet_sblock)
•             else:
•                 open_text_input = "Проверьте правильность ввода ключей"
•     elif cipher_choose_input == "Шифр Матричный":
•         input_matrix = list(map(int, matrix_input.split()))
•         matrix_input = [input_matrix[:3], input_matrix[3:6], input_matrix[6:]]
•         if matrix_input:
•             if matrix_check_parameters(matrix_input):
•                 if mode == "encrypt":
•                     cipher_text_input = matrix_encrypt(self.text_preparation(open_text_input), matrix_input,
alphabet)
•                 elif mode == "decrypt":
•                     open_text_input = matrix_decrypt(cipher_text_input, matrix_input, alphabet)
•         else:
•             if mode == "encrypt":

```

```

•         cipher_text_input = "Проверьте правильность ввода матрицы"
•     elif mode == "decrypt":
•         open_text_input = "Проверьте правильность ввода матрицы"
•
•     else:
•         if mode == "encrypt":
•             cipher_text_input = "Введите ключевую матрицу для шифра Матричный"
•             elif mode == "decrypt":
•                 open_text_input = "Введите ключевую матрицу для шифра Матричный"
• elif cipher_choose_input == "Шифр Плейфера":
•     keyword = self.keyword_edit.text()
•     if keyword:
•         if playfair_check_parameters(keyword, alphabet_playfair):
•             if mode == "encrypt":
•                 cipher_text_input = playfair_encrypt(self.text_preparation(open_text_input), keyword,
alphabet_playfair)
•             elif mode == "decrypt":
•                 open_text_input = playfair_decrypt(cipher_text_input, keyword, alphabet_playfair)
•         else:
•             if mode == "encrypt":
•                 cipher_text_input = "Проверьте правильность ключевого слова"
•             elif mode == "decrypt":
•                 open_text_input = "Проверьте правильность ключевого слова"
•         else:
•             if mode == "encrypt":
•                 cipher_text_input = "Введите ключевое слово для шифра Плейфера"
•             elif mode == "decrypt":
•                 open_text_input = "Введите ключевое слово для шифра Плейфера"
• elif cipher_choose_input == "Шифр вертикальной перестановки":
•     keyword = self.keyword_edit.text()
•     if keyword:
•         if vertical_transposition_check_parameters(keyword, alphabet):
•             if mode == "encrypt":

```

```

•         cipher_text_input = vertical_transposition_encrypt(self.text_preparation(open_text_input),
keyword, alphabet)
•
•         elif mode == "decrypt":
•             open_text_input = vertical_transposition_decrypt(cipher_text_input, keyword, alphabet)
•
•         else:
•             if mode == "encrypt":
•                 cipher_text_input = "Проверьте правильность ключевого слова"
•             elif mode == "decrypt":
•                 open_text_input = "Проверьте правильность ключевого слова"
•
•         else:
•             if mode == "encrypt":
•                 cipher_text_input = "Введите ключевое слово для шифра вертикальной перестановки"
•             elif mode == "decrypt":
•                 open_text_input = "Введите ключевое слово для шифра вертикальной перестановки"
•
•         elif cipher_choose_input == "Шифр решетки Кардано":
•             GridSizeInput = list(map(int, cardanosGridSizeInput.split(" ")))
•             GridCardano = [list(map(int, x.split(" "))) for x in cardanosGridGRIDInput.split(", ")]
•             if cardanosGridCheckParameters(GridSizeInput, GridCardano):
•                 if mode == "encrypt":
•                     cipher_text_input = cardanosGridEncrypt(self.text_preparation(open_text_input), GridSizeInput,
GridCardano, alphabet)
•                 elif mode == "decrypt":
•                     open_text_input = cardanosGridDecrypt(cipher_text_input, GridSizeInput, GridCardano, alphabet)
•
•             else:
•                 if mode == "encrypt":
•                     cipher_text_input = "Проверьте правильность ввода решетки"
•                 elif mode == "decrypt":
•                     open_text_input = "Проверьте правильность ввода решетки"
•
•         elif cipher_choose_input == "Шифр сеть Фейстель":
•             if mode == "encrypt":
•                 if feistelsNetworkCheckParameters(open_text_input, feistelsNetworkInput, alphabet_sblock):
•                     cipher_text_input = feistelsNetwork(open_text_input, feistelsNetworkInput, mode, alphabet_sblock)

```

```

•         else:
•             cipher_text_input = "Проверьте правильность ввода ключей"
•         elif mode == "decrypt":
•             if feistelsNetworkCheckParameters(cipher_text_input, feistelsNetworkInput, alphabet_sblock):
•                 open_text_input = feistelsNetwork(cipher_text_input, feistelsNetworkInput, mode, alphabet_sblock)
•             else:
•                 open_text_input = "Проверьте правильность ввода ключей"
•         elif cipher_choose_input == "Одноразовый блокнот Шеннона":
•             if mode == "encrypt":
•                 if shannonsNotebookCheckParameters(*list(map(int, shannonsNotebookT_A_C.split())), alphabet):
•                     cipher_text_input = shannonsNotebookEncrypt(self.text_preparation(open_text_input), *list(map(int,
shannonsNotebookT_A_C.split()))), alphabet)
•                 else:
•                     cipher_text_input = "Проверьте правильность ввода ключей"
•             elif mode == "decrypt":
•                 if shannonsNotebookCheckParameters(*list(map(int, shannonsNotebookT_A_C.split())), alphabet):
•                     open_text_input = shannonsNotebookDecrypt(open_text_input, *list(map(int,
shannonsNotebookT_A_C.split()))), alphabet)
•                 else:
•                     open_text_input = "Проверьте правильность ввода ключей"
•         elif cipher_choose_input == "Hexoize":
•             if mode == "encrypt":
•                 cipher_text_input = StrToHex(open_text_input)
•             elif mode == "decrypt":
•                 open_text_input = HexToStr(cipher_text_input)
•         elif cipher_choose_input == "Магма (гаммирование)":
•             if mode == "encrypt":
•                 if MagmaCheckParameters(magmaKeyInput, magmaVectorInput, alphabet_sblock):
•                     cipher_text_input = magma(open_text_input, magmaKeyInput, magmaVectorInput, mode, alphabet_sblock)
•                 else:
•                     cipher_text_input = "Проверьте правильность ввода ключей"
•             elif mode == "decrypt":

```

```
•         if MagmaCheckParameters(magmaKeyInput, magmaVectorInput, alphabet_sblock):
•             open_text_input = magma(cipher_text_input, magmaKeyInput, magmaVectorInput, mode, alphabet_sblock)
•         else:
•             open_text_input = "Проверьте правильность ввода ключей"
•     else:
•         pass
•
•     # Обновление текста в виджетах
•     self.open_text_edit.setPlainText(open_text_input)
•     self.cipher_text_edit.setPlainText(cipher_text_input)
•
• if __name__ == '__main__':
•     app = QApplication(sys.argv)
•     ex = CipherApp()
•     ex.show()
•     sys.exit(app.exec_())
```


8. Одноразовый блокнот К.Шеннона

Открытый текст сообщения m записывают, как последовательность бит или символов $m = m_0m_1\dots m_{n-1}$, а двоичную или символьную шифрующую последовательность k той же самой длины — как $k = k_0k_1\dots k_{n-1}$.

Шифртекст $c = c_0c_1\dots c_{n-1}$ определяется соотношением $c_i = m_i \oplus k_i$ при $0 \leq i \leq n-1$, где \oplus обозначает операцию «исключающее ИЛИ» (ассемблерная операция XOR) по модулю два или сложение по любому другому модулю в случае символьной гаммы.

Блок-схема программы

Код программы с комментариями

```
def gcd(a, b):
    if a == 0:
        return b
    return gcd(b % a, a)

def shannonsNotebookCheckParameters(t, a, c, alphabet):
    if not (t and a and c):
        return False # t, a и c не являются числами
    if not (t > 0 and a > 0 and c > 0):
        return False # Числа отрицательные
    if t > 31 or a > 31 or c > 31:
        return False # Числа больше или равны модулю
    if a % 4 != 1:
        return False # a по модулю 4 не равно 1
    if gcd(c, 32) != 1:
        return False # c не соизмеримо с t
    return True

def shannonsNotebookEncrypt(openText, t, a, c, alphabet):
    for letter in openText:
        if letter not in alphabet:
            return "Введённый текст содержит запрещённые символы" # Буквы текста не содержатся в алфавите
    encryptedText = "" # Шифртекст
    gamma = [t]
    # Создание гаммы длины openText
    for i in range(len(openText)):
        gamma.append((a * gamma[-1] + c) % len(alphabet))
    # xor гаммы и открытого текста
    for i in range(len(openText)):
        encryptedText += str((alphabet.index(openText[i]) + 1) ^ gamma[i]).zfill(2)[-2:]
```

```

    return encryptedText # Возврат шифртекста

def shannonsNotebookDecrypt(encryptedText, t, a, c, alphabet):
    decryptedText = "" # Расшифрованный текст
    gamma = [t]
    # Создание гаммы длины encryptedText / 2
    for i in range(len(encryptedText) // 2):
        gamma.append((a * gamma[-1] + c) % len(alphabet))
    encryptedTextArr = [encryptedText[i:i+2] for i in range(0, len(encryptedText), 2)]
    # Проверка наличия значений для расшифровки
    if not encryptedTextArr:
        return ""
    # xor гаммы и шифртекста
    for i in range(len(encryptedTextArr)):
        try:
            decrypted_char_index = (int(encryptedTextArr[i]) ^ gamma[i]) - 1
            decrypted_char = alphabet[decrypted_char_index % len(alphabet)]
            decryptedText += decrypted_char
        except ValueError:
            # Обработка случая, когда строка не может быть преобразована в целое число
            pass
    decryptedText = decryptedText.replace("тчк", ".").replace("зпт", ",").replace("тире", "-").replace('прбл', ' ')
    decryptedText = decryptedText.replace('двтч', ':').replace('тчсзп', ';').replace('отскб', '(').replace('зксб', ')').replace('впрзн', '?')
    decryptedText = decryptedText.replace('восклзн', '!').replace('првст', '\n')
    return decryptedText # Возврат расшифрованного текста

```

Тестирование

Шифры

Выберите шифр: Одноразовый блокнот Шеннона

Введите открытый текст(Расшифрованный):

Тот, кто ложится на два стула, падает на ребра.

Шифрованный текст:

240126202325141112101419221715482131281522080628312325202802120510082618062703261807181220

Введите сдвиг для шифра Цезаря

Введите ключевое слово для шифра Белазо, Плейфера, Вертикальной перестановки

Введите ключевую букву для шифра Виженера

Введите ключевую матрицу для шифра Матричный

Введите размерность решётки для шифра Кардано (8 8)

Решётка (0 1, 1 6, 2 1, 2 2, 2 4, 3 6, 3 7, 4 3, 4 5, 5 0, 6 3, 6 5, 6 7, 7 2, 7 4, 7 7)

Ключ для шифра сети Фейстеля

11 9 11

Ключ для Магна(гаммирования)

Инициализирующий вектор для Магна(гаммирования)

Выберите режим: Шифрование

Выполнить

☐ Расширенный текст

```
create_histogram("Тот3ПТктоложитс2надвастула3ПТпадаетнаребраТЧК", 1, "Открытый текст")

alp = list(map(str, list(range(1, 32))))
for i in range(len(alp)):
    while len(alp[i]) != 2:
        alp[i] = '0' + alp[i]
alp = ' '.join(alp)
create_histogram("24 01 26 20 23 25 14 11 12 10 14 19 22 17 15 48 21 31 28 15 22 02 06 28 31 23 25 20 28 02 12 05 10 08 26 18 06 27 03 26 18 07 18 12 20", 2, "одноразовый блокнот шеннона", alp)
```

Работа с текстом не менее 1000 знаков

Зашифрование

Шифры

Выберите шифр:

Одноразовый блокнот Шеннона

Введите открытый текст (Расшифрованный):

Жизнь - это удивительное приключение, полное разнообразных событий и встреч. В каждом моменте мы находим что-то новое и уникальное. Стремление к росту и саморазвитию вдохновляет нас на поиск новых горизонтов. Важно помнить, что каждый шаг вперед приносит с собой уроки и опыт.

Разнообразие культур, языков и традиций делает наш мир удивительно богатым. Общение с людьми разных национальностей расширяет кругозор, позволяя нам понимать и уважать друг друга. Взаимное уважение и терпимость создают основу для гармоничного сосуществования.

Природа тоже играет важную роль в нашей жизни. Красота закатов, шум океана, пение птиц - все это напоминает нам о величии мира природы. Забота о окружающей среде становится частью нашей ответственности.

Шифрованный текст:

1207011826262026152108052518121823001003071123040719240706011112220006262327072019230831192905222123310407090525172601110328020627311116221113221209030530101912131408142708261015052414310315120230243011092311180025070709130125182628062425031817201102022802070306170104222301923032418301301251823130121160130060006031030152407242620052109011015272813270010282105071123040724241514301614040826041226202615201805251517222123312806242503030719083029311931302430110321090110192118131217192916313005052517261801000403150802412620262026150800061519312821310929212205072604262030160114041321102327072007091629231319032029102028100410312116030129291709022519100412113014170715193128031322280624250318172024050228020723082423270720002207052507130125180929302026262605012113303116261205140808101819230324111918271814081427190525172630201901302400302431212521090020180530101931202808133119192402202915273016193131011220262026153809082013300010282105071123040007160106050410273111160212092505211705291419170214081427232825030719082723040302291516260410081804132216062803071910202810041031251930092804062731111621126202615103017021520001028212922182720050600060328122302010728112311231721073015193128103111213011531400060006

Введите сдвиг для шифра Цезаря

Введите ключевое слово для шифра Белазо, Плейфера, Вертикальной перестановки

Введите ключевую букву для шифра Виженера

Введите ключевую матрицу для шифра Матричный

Введите размерность решётки для шифра Кардано (8 8)

Решётка (0 1, 1 6, 2 1, 2 2, 2 4, 3 6, 3 7, 4 3, 4 5, 5 0, 6 3, 6 5, 6 7, 7 2, 7 4, 7 7)

Ключ для шифра сети Фейстеля

11 9 11

Ключ для Магма(гаммирования)

Инициализирующий вектор для Магма(гаммирования)

Выберите режим:

Шифрование

Выполнить

☒ Расширенный текст

Расшифрование

Шифры

Выберите шифр:

Одноразовый блокнот Шеннона

Введите открытый текст (Расшифрованный):

жизнь - это удивительное приключение, полное разнообразных событий и встреч. в каждом моменте мы находим что-то новое и уникальное. стремление к росту и саморазвитию вдохновляет нас на поиск новых горизонтов. важно помнить, что каждый шаг вперед приносит с собой уроки и опыт.

разнообразие культур, языков и традиций делает наш мир удивительно богатым. общение с людьми разных национальностей расширяет кругозор, позволяя нам понимать и уважать друг друга. взаимное уважение и терпимость создают основу для гармоничного сосуществования.

природа тоже играет важную роль в нашей жизни. красота закатов, шум океана, пение птиц - все это напоминает нам о величии мира природы. забота о окружающей среде становится неотъемлемой частью ответственного образа жизни.

Шифрованный текст:

031819230324241121301813010707112304282028020102280207072715082712561415090412181218232931301722091403071908312931091408052314262026151517052914133130273129161720240220290731031512243129250922030818041325161527300802092921220707271916272802280207291512113026081804132112151802142408240711230416301613022929150507042909033708180413291519312823012715300913030307182228022803252926291203080818041327291928242019172804100410312416291402280207090020092321090110182914041301251807312410041031273012142706012731111622111320103015080918121823192201172006071630012331031512250115251404032213262304140017192024280525210616240600060317120705012513050618301315180915193128163130091116052517262811022003192731111614262026152018052515200306125728062425132126162318022802070300132327072000091005110613012518231305100410315400280902280207072513050609122731070707091301251811280624250412203007181305162430243011250309190015292515130125181628062425312126231629012806060215180905231130251705291416121115270026211814300600060300020814151029202321090110192316291301251813023017200414240527310315120228261922032616192117052914191711172102172027281206000603000028250206172327072010221622191928212012101708091303

Введите сдвиг для шифра Цезаря

Введите ключевое слово для шифра Белазо, Плейфера, Вертикальной перестановки

Введите ключевую букву для шифра Виженера

Введите ключевую матрицу для шифра Матричный

Введите размерность решётки для шифра Кардано (8 8)

Решётка (0 1, 1 6, 2 1, 2 2, 2 4, 3 6, 3 7, 4 3, 4 5, 5 0, 6 3, 6 5, 6 7, 7 2, 7 4, 7 7)

Ключ для шифра сети Фейстеля

11911

Ключ для Магна(гаммирования)

Инициализирующий вектор для Магна(гаммирования)

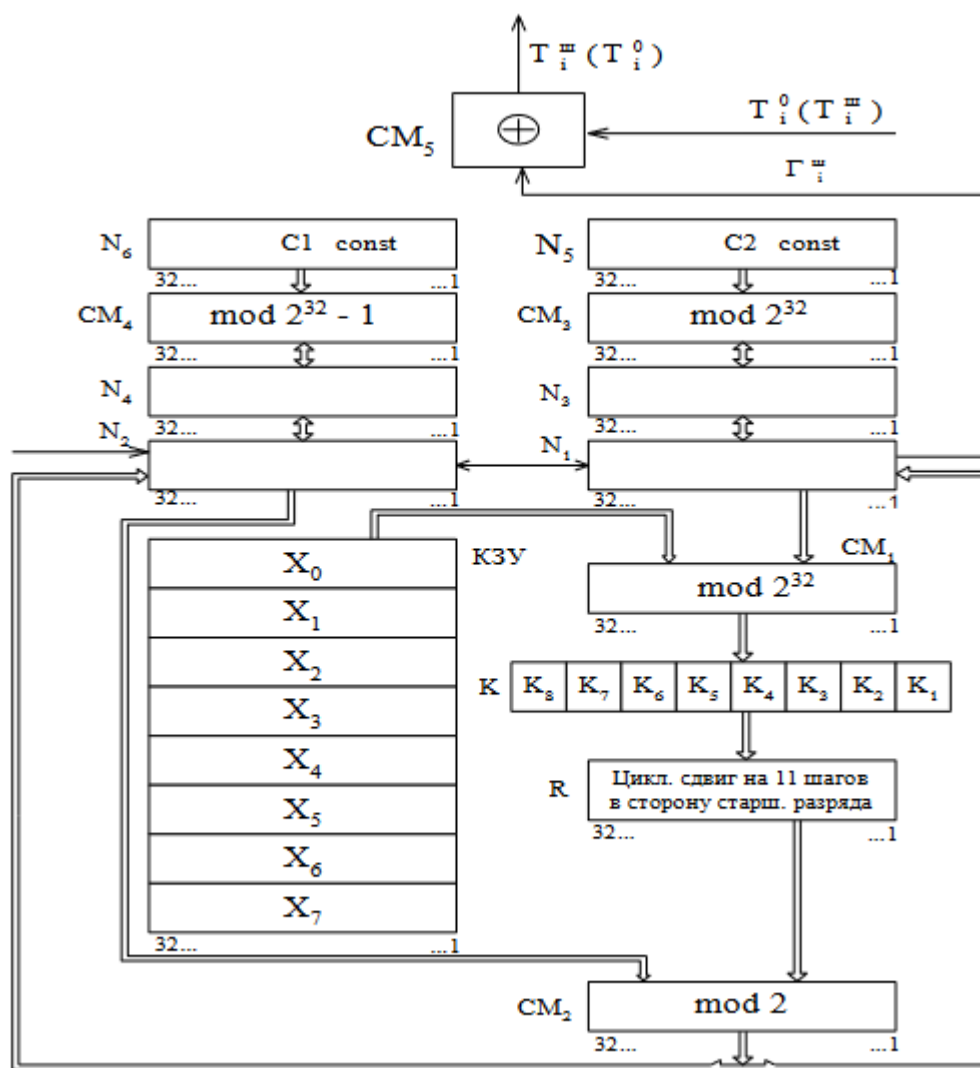
Выберите режим:

Расшифрование

Выполнить

☒ Расширенный текст

14. Гаммирование ГОСТ 28147-89 и ГОСТ Р 34.13-2015 (ГОСТ Р 34.12-2015 «Магма»)



Блок-схема программы

Код программы с комментариями

```
from feistelsNetwork import feistelsNetworkCheckParameters, feistelsNetwork

def MagmaCheckParameters(key, init_vector, alphabet):
    for digit in init_vector:
        if not digit.isdigit():
            return False
    for letter in key:
        if letter not in alphabet:
            return False
    return True

def magma(open_text, key, init_vector, mode, alphabet):
    def xor_hex(hex1, hex2):
        binary1 = int(hex1, 16)
        binary2 = int(hex2, 16)
        xor_result = (binary1 ^ binary2).to_bytes((max(binary1.bit_length(), binary2.bit_length()) + 7) // 8,
byteorder='big').hex()
        return xor_result

    encrypted_text = ""
    init_vector = (init_vector + "0000000000000000")[:16]
    keys = [key[i:i+8] for i in range(0, len(key), 8)]
    rev_keys = keys = keys[::-1]
    keys = keys + keys + keys
    keys = keys + rev_keys

    total = ""
    for i in range(0, len(open_text), 16):
        p_i = open_text[i:i+16]
        a0 = init_vector[:8]
        a1 = init_vector[8:16]
```



```
ek = feistelsNetwork(init_vector, key, "encrypt", [])
c_i = xor_hex(ek, p_i)
total += c_i
init_vector = hex(int(init_vector, 16) + 1)[2:]

encrypted_text = total
return encrypted_text
```

Тестирование

Шифры

Выберите шифр:

Магма (гаммирование)

Введите открытый текст (Расшифрованный):

42243e44202c02043a44243e02043b43e43643844244144f02043d43002043443243002044144244343b43002c02043f43043443043544202043d43002044043543144043002e

Шифрованный текст:

9e62df238b88b161df35eea92cf4fb0abdc33e22225ecdccdd98cb58bcc79c283100057e03a77142c6f7cf856ef5cb52a0efe1cfc87b119c4a728369dd7df61b2adddf68abff4c4f

Введите сдвиг для шифра Цезаря

Введите ключевое слово для шифра Белазо, Плейфера, Вертикальной перестановки

Введите ключевую букву для шифра Виженера

Введите ключевую матрицу для шифра Матричный

Введите размерность решётки для шифра Кардано (8 8)

Решётка (0 1, 1 6, 2 1, 2 2, 2 4, 3 6, 3 7, 4 3, 4 5, 5 0, 6 3, 6 5, 6 7, 7 2, 7 4, 7 7)

Ключ для шифра сети Фейстеля

т а с для Одноразового блокнота Шеннона

ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff

12345678

Выберите режим:

Шифрование

Выполнить

☐ Расширенный текст