Федеральное государственное автономное

образовательное учреждение

высшего профессионального образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

институт

Прикладная математика и компьютерная безопасность

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 17**

по Криптографические методы защиты информации

наименование дисциплины

Программная реализация шифра ГОСТ 28147-89

тема

Преподаватель В.И. Вайнштейн

подпись**,** дата инициалы, фамилия

Студент КИ17-01, 031722011 К.А. Василенко

номер группы, зачетной книжки подпись**,** дата инициалы, фамилия

Красноярск 2021

# ВВЕДЕНИЕ

Цель работы: реализовать шифр ГОСТ 28147-89 (шифрование и расшифрование) на любом языке программирования. Предусмотреть графический интерфейс.

Задачи:

1. реализовать программно шифр ГОСТ 28147-89, предусматривая графический интерфейс;
2. провести тесты на работоспособность программы;
3. сделать отчёт о проделанной работе.

**Реализация шифра**

1. **Описание шифра**

**ГОСТ 28147-89** «*Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования*» — устаревший государственный стандарт союза ССР (а позже межгосударственный стандарт СНГ), описывающий алгоритм [симметричного](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) блочного шифрования и режимы его работы.

**Режим простой замены**

Для зашифровывания в этом режиме 64-битный блок открытого текста сначала разбивается на две половины: Tо = (A0, B0). На i-ом цикле используется подключ Xi:

Ai+1 = Bi xor f(Ai, Xi)

Bi+1 = Ai

Для генерации подключей исходный 256-битный ключ разбивается на восемь 32-битных чисел: K0…K7.

Подключи X0…X23 являются циклическим повторением K0…K7. Подключи X24…X31 являются K7…K0.

Результатом выполнения всех 32 раундов алгоритма является 64-битный блок шифртекста: Tш = (A32, B32).

Расшифрование осуществляется по тому же алгоритму, что и зашифрование, с тем изменением, что инвертируется порядок подключей: X0…X7 являются K0…K7, а X8…X31 являются циклическим повторением K7…K0.

Во входных и выходных данных 32-битные числа представляются как little endian.

**Функция** f(Ai, Xi) **f ( A i , X i ) {\displaystyle f(A\_{i},X\_{i})}** вычисляется следующим образом:

Ai и Xi складываются по модулю 232.

Результат разбивается на восемь 4-битовых подпоследовательностей, каждая из которых поступает на вход своего *узла таблицы замен* (в порядке возрастания старшинства битов), называемого ниже *S-блоком*. Общее количество S-блоков стандарта — восемь, то есть столько же, сколько и подпоследовательностей. Каждый *S-блок* представляет собой перестановку чисел от 0 до 15 (конкретный вид S-блоков в стандарте не определен). Первая 4-битная подпоследовательность попадает на вход первого S-блока, вторая — на вход второго и т. д.

Если узел *S-блока* выглядит так:

1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12

и на входе S-блока 0, то на выходе будет 1, если 4, то на выходе будет 5, если на входе 12, то на выходе 6 и т. д.

Выходы всех восьми S-блоков объединяются в 32-битное слово, затем всё слово циклически сдвигается влево (к старшим разрядам) на 11 битов.

Режим простой замены имеет следующие недостатки:

* Может применяться только для шифрования открытых текстов с длиной, кратной 64 бит
* При шифровании одинаковых блоков открытого текста получаются одинаковые блоки шифротекста, что может дать определенную информацию криптоаналитику.

Таким образом, применение ГОСТ 28147-89 в режиме простой замены желательно лишь для шифрования ключевых данных.

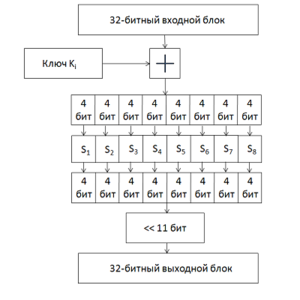


Рисунок 1 - Функция f(Ai, Xi), используемая в сети Фейстеля.

**Гаммирование с обратной связью**

Алгоритм шифрования похож на режим гаммирования, однако гамма формируется на основе предыдущего блока зашифрованных данных, так что результат шифрования текущего блока зависит также и от предыдущих блоков. По этой причине данный режим работы также называют гаммированием с зацеплением блоков.

Алгоритм шифрования следующий:

1. Синхропосылка заносится в регистры N1 и N2
2. Содержимое регистров N1 и N2 шифруется в соответствии с алгоритмом простой замены. Полученный результат является 64-битным блоком гаммы.
3. Блок гаммы побитно складывается по модулю 2 с блоком открытого текста. Полученный шифротекст заносится в регистры N1 и N2
4. Операции 2-3 выполняются для оставшихся блоков требующего шифрования текста.

При изменении одного бита шифротекста, полученного с использованием алгоритма гаммирования с обратной связью, в соответствующем блоке расшифрованного текста меняется только один бит, так же затрагивается последующий блок открытого текста. При этом все остальные блоки остаются неизменными.

При использовании данного режима следует иметь в виду, что синхропосылку нельзя использовать повторно (например, при шифровании логически раздельных блоков информации - сетевых пакетов, секторов жёсткого диска и т. п). Это обусловлено тем, что первый блок шифр-текста получен всего лишь сложением по модулю два с зашифрованной синхропосылкой; таким образом, знание всего лишь 8 первых байт исходного и шифрованного текста позволяют читать первые 8 байт любого другого шифр-текста после повторного использования синхропосылки.

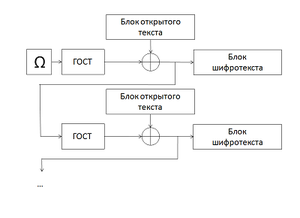


Рисунок 2 - Схема работы в режиме гаммирования с обратной связью.

{\displaystyle c\_{j}=(m\_{j}+k\_{j})\mod {n}}{\displaystyle m\_{j}=(c\_{j}+n-k\_{j})\mod {n}}**Программная реализация**

Программа написана на языке Python с использованием библиотеки PyQt5 и Qt designer для отрисовки графического интерфейса.

1. **Листинг с описанием основного алгоритма программы**

**gost.py**

**from** PyQt5 **import** QtWidgets  
**from** tkinter **import** Tk, Label  
**import** time  
  
#возвращает массив из 8 элементов, где каждый элемент - это 32 бита  
**def** genKeys\_GOST(key256b:str):  
 keys1 = key256b.encode()  
 keys = []  
 **for** i **in** range(8):  
 keys.append(keys1[:4])  
 keys1 = keys1[4:]  
 keysBin = []  
 **for** i **in** range(len(keys)):  
 binStr = ''  
 **for** j **in** keys[i]:  
 binStr += bin(j)[2:].zfill(8)  
 keysBin.append(binStr)  
 **return** keysBin  
  
**def** before\_coding(text, flag: int):  
 file = 0  
 **if** type(text) == str:  
 **if** flag == 1:  
 textToInt = text.encode()  
 **else**:  
 textToInt = b''  
 **for** i **in** text:  
 textToInt += bytes([ord(i)])  
 **else**:  
 textToInt = text  
 file = 1  
 print('OK')  
  
 blocks = []  
 **while** len(textToInt) > 0:  
 blocks.append(textToInt[:8])  
 textToInt = textToInt[8:]  
  
 count = 0  
 **if** len(blocks[len(blocks) - 1]) != 8:  
 **while** len(blocks[len(blocks) - 1]) != 8:  
 count += 1  
 blocks[len(blocks) - 1] += bytes([0])  
 blocks[len(blocks) - 1] = blocks[len(blocks) - 1][:-1] + bytes([count])  
  
 blocksBin = []  
 **for** i **in** range(len(blocks)):  
 binStr = ''  
 **for** j **in** blocks[i]:  
 binStr += bin(j)[2:].zfill(8)  
 blocksBin.append(binStr)  
 **return** blocksBin, file  
  
**def** after\_coding(file, block64Int, flag):  
 **if** file == 0:  
 **if** flag != 1:  
 numb = block64Int[len(block64Int) - 1]  
 print('n', numb)  
 blocksText1 = block64Int  
 j = 2  
 **for** i **in** range(numb-1):  
 **if** blocksText1[-j] == 0:  
 blocksText1 = blocksText1[:-1]  
 j = 1  
 **else**:  
 **break  
 if** len(blocksText1) < len(block64Int):  
 blocksText1 = blocksText1[:-1]  
 blocksText = ''  
 **for** i **in** blocksText1:  
 blocksText += chr(i)  
 **else**:  
 blocksText = ''  
 **for** i **in** block64Int:  
 blocksText += chr(i)  
 **else**:  
 **if** flag != 1:  
 numb = block64Int[len(block64Int) - 1]  
 print('n', numb)  
 blocksText = block64Int  
 j = 2  
 **for** i **in** range(numb-1):  
 **if** blocksText[-j] == 0:  
 blocksText = blocksText[:-1]  
 j = 1  
 **else**:  
 **break  
 if** len(blocksText) < len(block64Int):  
 blocksText = blocksText[:-1]  
 **else**:  
 blocksText = block64Int  
 **return** blocksText  
  
**def** xor(a: str, b: str):  
 **if** len(a) > len(b):  
 a = a.zfill(len(b))  
 **elif** len(b) > len(a):  
 b = b.zfill(len(a))  
 res = ""  
 **for** i **in** range(len(a)):  
 **if** a[i] == '1':  
 **if** b[i] == '1':  
 res += '0'  
 **else**:  
 res += '1'  
 **if** a[i] == '0':  
 **if** b[i] == '1':  
 res += '1'  
 **else**:  
 res += '0'  
 **return** res  
  
**def** sdvig(str: str, n: int):  
 arr = []  
 **for** i **in** range(len(str)):  
 arr.append('')  
 **for** i **in** range(len(arr)):  
 arr[(i - n) % len(arr)] = str[i]  
 arrTxt = ''  
 **for** i **in** arr:  
 arrTxt += i  
 **return** arrTxt  
  
**def** substitution(N1:str):  
 \_\_Sbox = [  
 [9, 6, 3, 2, 8, 11, 1, 7, 10, 4, 14, 15, 12, 0, 13, 5],  
 [3, 7, 14, 9, 8, 10, 15, 0, 5, 2, 6, 12, 11, 4, 13, 1],  
 [14, 4, 6, 2, 11, 3, 13, 8, 12, 15, 5, 10, 0, 7, 1, 9],  
 [14, 7, 10, 12, 13, 1, 3, 9, 0, 2, 11, 4, 15, 8, 5, 6],  
 [11, 5, 1, 9, 8, 13, 15, 0, 14, 4, 2, 3, 12, 7, 10, 6],  
 [3, 10, 13, 12, 1, 2, 0, 11, 7, 5, 9, 4, 8, 15, 14, 6],  
 [1, 13, 2, 9, 7, 10, 6, 0, 8, 12, 4, 5, 15, 3, 11, 14],  
 [11, 10, 15, 5, 0, 12, 14, 8, 6, 2, 3, 9, 1, 7, 13, 4]  
 ]  
 # 8 блоков по 4 бита  
 blocks4b = []  
 **for** i **in** range(8):  
 blocks4b.append(N1[:4])  
 N1 = N1[4:].zfill(4)  
 blocksAfterSbox = ''  
 **for** i **in** range(8):  
 blocksAfterSbox+=bin(\_\_Sbox[i][int(blocks4b[i], 2)])[2:].zfill(4)  
 **return** sdvig(blocksAfterSbox, 11)  
  
**def** round\_feistel\_scheme(L0: str, R0: str, key: str):  
 # RES = (N1 + Ki) mod 2 ^ 32  
 RES = bin((int(L0, 2) | int(key, 2)) % 2\*\*32)[2:]  
  
 # RES = RES -> Sbox, << 11  
 RES = substitution(RES)  
 L0, R0 = xor(RES, R0), L0  
 **return** L0, R0  
  
**def** feistel\_scheme(block:str, keys:list, flag:int):  
 L0 = block[:32]  
 R0 = block[32:]  
 **if** flag == 1:  
 # K0, K1, K2, K3, K4, K5, K6, K7, K0, K1, K2, K3, K4, K5, K6, K7, K0, K1, K2, K3, K4, K5, K6, K7  
 **for** i **in** range(3):  
 **for** j **in** range(len(keys)):  
 L0, R0 = round\_feistel\_scheme(L0, R0, keys[j])  
 # K7, K6, K5, K4, K3, K2, K1, K0  
 **for** i **in** range(len(keys)):  
 L0, R0 = round\_feistel\_scheme(L0, R0, keys[len(keys) - 1 - i])  
 L0, R0 = R0, L0  
 **else**:  
 # K0, K1, K2, K3, K4, K5, K6, K7  
 **for** i **in** range(len(keys)):  
 L0, R0 = round\_feistel\_scheme(L0, R0, keys[i])  
 # K7, K6, K5, K4, K3, K2, K1, K0, K7, K6, K5, K4, K3, K2, K1, K0, K7, K6, K5, K4, K3, K2, K1, K0  
 **for** i **in** range(3):  
 **for** j **in** range(len(keys)):  
 L0, R0 = round\_feistel\_scheme(L0, R0, keys[len(keys) - 1 - j])  
 L0, R0 = R0, L0  
 **return** L0 + R0  
  
**def** GOST\_zamena(keys1, text, flag):  
 start = time.time()  
 **if not** keys1 **or** keys1 == '':  
 keys = genKeys\_GOST('this\_is\_a\_pasw\_for\_GOST\_28147\_89')  
 **else**:  
 **if** len(keys1) != 256:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 **for** i **in** keys1:  
 **if** (i != '1' **and** i != 1 **and** i != '0' **and** i != 0) == **True**:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 keys = [0, 0, 0, 0, 0, 0, 0, 0]  
 **for** i **in** range(8):  
 keys[i] = keys1[:32]  
 keys1 = keys1[32:]  
  
 **if not** text:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Введите текст!")  
 msgBox.exec\_()  
 **return** ("")  
  
 blocksBin, file = before\_coding(text, flag)  
  
 percent = 100 / len(blocksBin)  
 progress = 0  
 normal = 0  
 root = Tk()  
  
 label = Label(text='Progressbar')  
 x = (root.winfo\_screenwidth() - root.winfo\_reqwidth()) / 2  
 y = (root.winfo\_screenheight() - root.winfo\_reqheight()) / 2  
 root.wm\_geometry("+%d+%d" % (x, y))  
 label.pack()  
  
 block64Int = b''  
 **for** i **in** range(len(blocksBin)):  
 block64b = feistel\_scheme(blocksBin[i], keys, flag)  
 **try**:  
 progress += percent  
 label.config(text='**\r**Обработка завершена на %3d%%' % progress)  
 root.update()  
 #print('\rОбработка завершена на %3d%%' % progress, end = '', flush = True)  
 time.sleep(0.01)  
 **except** Exception:  
 normal = 1  
 **pass  
 for** j **in** range(8):  
 block64Int+=bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
  
 blocksText = after\_coding(file, block64Int, flag)  
 **if** normal == 0:  
 root.destroy()  
 print('Время выполнения: ', (time.time() - start) / 60)  
 **return** blocksText  
  
**def** GOST\_gamm(keys1, text, flag, vector\_init):  
 start = time.time()  
 **if not** keys1 **or** keys1 == '':  
 keys = genKeys\_GOST('this\_is\_a\_pasw\_for\_GOST\_28147\_89')  
 **else**:  
 **if** len(keys1) != 256:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 **for** i **in** keys1:  
 **if** (i != '1' **and** i != 1 **and** i != '0' **and** i != 0) == **True**:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 keys = [0, 0, 0, 0, 0, 0, 0, 0]  
 **for** i **in** range(8):  
 keys[i] = keys1[:32]  
 keys1 = keys1[32:]  
 **if not** vector\_init **or** vector\_init == '':  
 vector\_init = '1110011110010101110010111001010010000111001010011100101001001011'  
 **else**:  
 **if** len(vector\_init) != 64:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в векторе инициализации!")  
 msgBox.exec\_()  
 **return** ("")  
 **for** i **in** vector\_init:  
 **if** (i != '1' **and** i != 1 **and** i != '0' **and** i != 0) == **True**:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в векторе инициализации!!")  
 msgBox.exec\_()  
 **return** ("")  
 **if not** text:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Введите текст!")  
 msgBox.exec\_()  
 **return** ("")  
  
 blocksBin, file = before\_coding(text, flag)  
  
 percent = 100 / len(blocksBin)  
 progress = 0  
 normal = 0  
 root = Tk()  
  
 label = Label(text='Progressbar')  
 x = (root.winfo\_screenwidth() - root.winfo\_reqwidth()) / 2  
 y = (root.winfo\_screenheight() - root.winfo\_reqheight()) / 2  
 root.wm\_geometry("+%d+%d" % (x, y))  
 label.pack()  
  
 **if** flag == 1:  
 N1 = vector\_init[:32]  
 N2 = vector\_init[32:]  
 block64Int = b''  
 **for** i **in** range(len(blocksBin)):  
 gamma = feistel\_scheme(N1 + N2, keys, 1)  
 **try**:  
 progress += percent  
 label.config(text='**\r**Обработка завершена на %3d%%' % progress)  
 root.update()  
 # print('\rОбработка завершена на %3d%%' % progress, end = '', flush = True)  
 time.sleep(0.01)  
 **except** Exception:  
 normal = 1  
 **pass** block64b = xor(gamma, blocksBin[i])  
 N1 = block64b[:32]  
 N2 = block64b[32:]  
 **for** j **in** range(8):  
 block64Int += bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
 **else**:  
 N = vector\_init  
 block64Int = b''  
 **for** i **in** range(len(blocksBin)):  
 gamma = feistel\_scheme(N, keys, 1)  
 **try**:  
 progress += percent  
 label.config(text='**\r**Обработка завершена на %3d%%' % progress)  
 root.update()  
 # print('\rОбработка завершена на %3d%%' % progress, end = '', flush = True)  
 time.sleep(0.01)  
 **except** Exception:  
 normal = 1  
 **pass** block64b = xor(gamma, blocksBin[i])  
 **for** j **in** range(8):  
 block64Int += bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
 N = blocksBin[i]  
  
 blocksText = after\_coding(file, block64Int, flag)  
 **if** normal == 0:  
 root.destroy()  
 print('Время выполнения: ', (time.time() - start) / 60)  
 **return** blocksText

1. **Примеры работы программы**

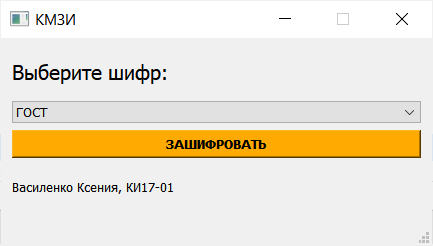


Рисунок 3 – Главное окно.

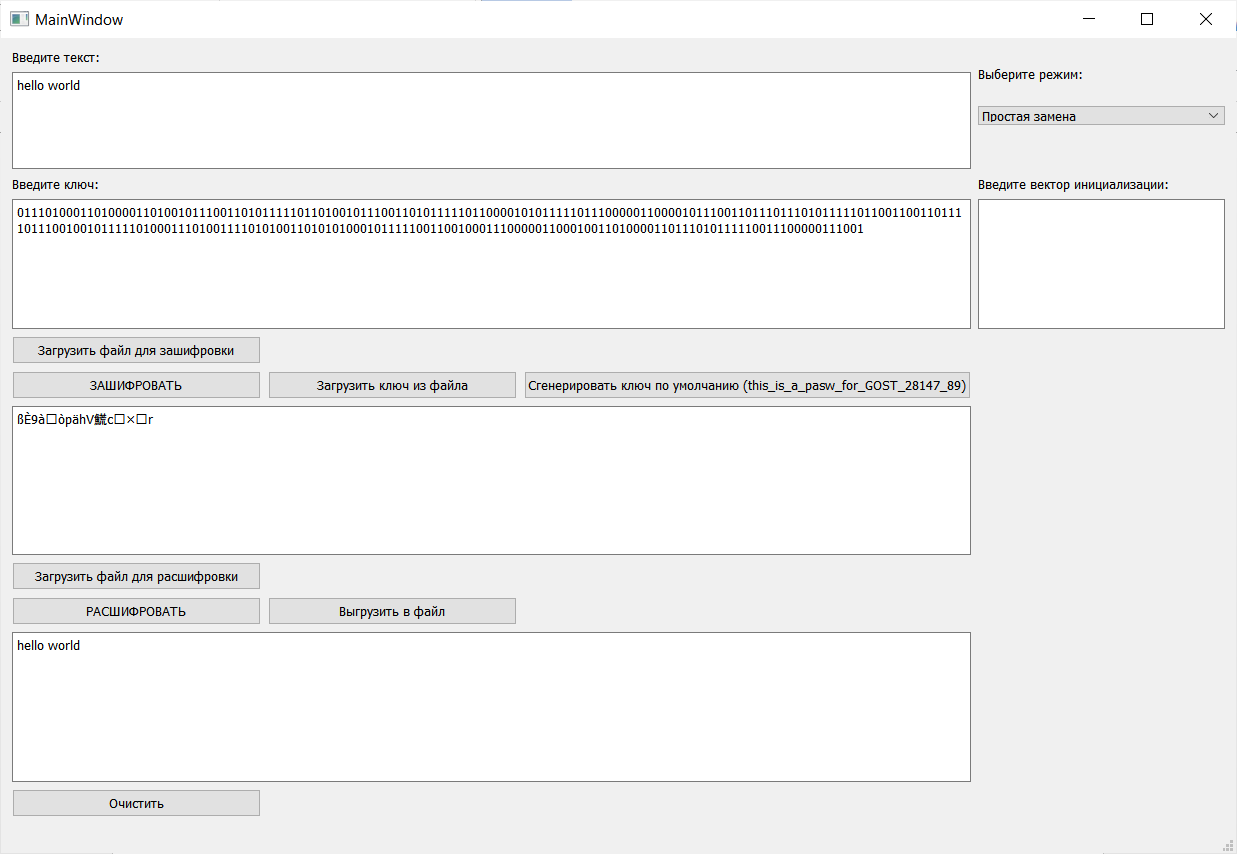


Рисунок 4 – Шифр ГОСТ – Простая замена.

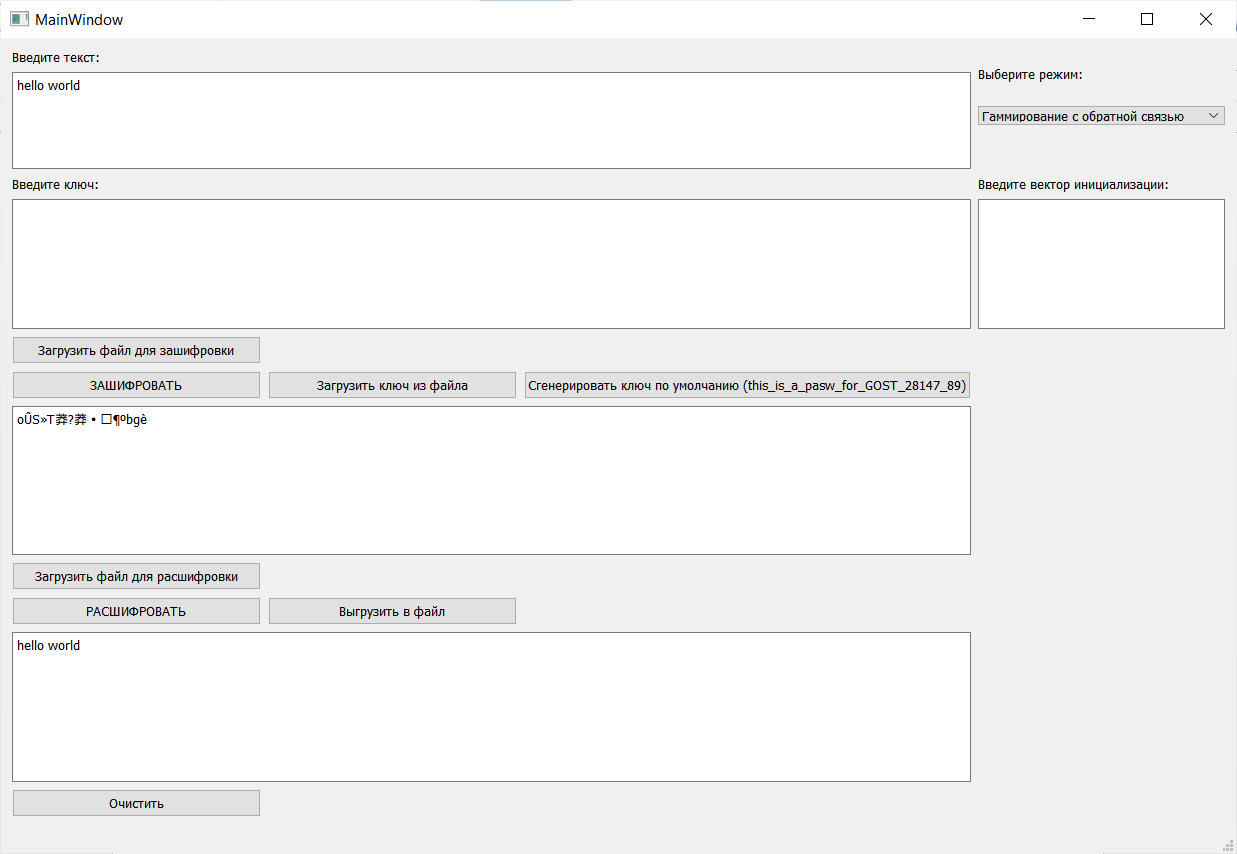
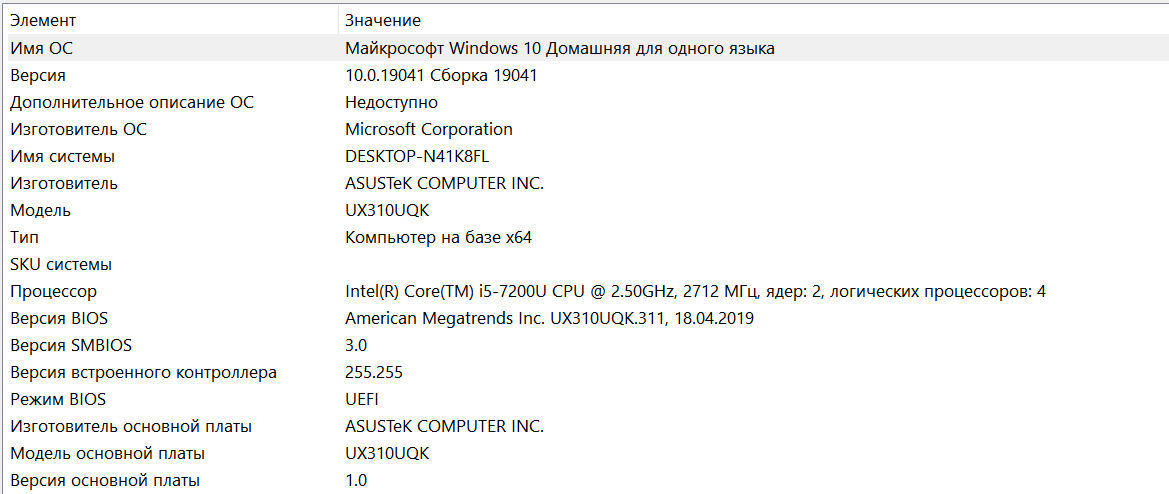


Рисунок 5 – Шифр ГОСТ – Гаммирование с обратной связью.

1. **Тестирование**

Характеристики:



Файл 2 МБ обрабатывается (и зашифровка, и расшифровка) в режиме простой замены примерно 1,5 часа.

# ЗАКЛЮЧЕНИЕ

Во время выполнения лабораторной работы №17 мною был изучен и реализован программно шифр ГОСТ 28147-89, а также предусмотрен графический интерфейс с помощью PyQt5 и Qt designer.