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 7. КОМБИНАЦИОННЫЕ ШИФРЫ

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.

17.МАГМА

Шифр Магма - симметричный блочный шифр, работает на основе принципа подстановки и перестановки битов данных в блоках. Методом применения ключа к исходному тексту выполняются последовательные перестановки и замены битов, что приводит к зашифрованию сообщения. Этот процесс повторяется для каждого блока данных в тексте.

Блок-схема программы

Режим простой замены

Режим гаммирования

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
```

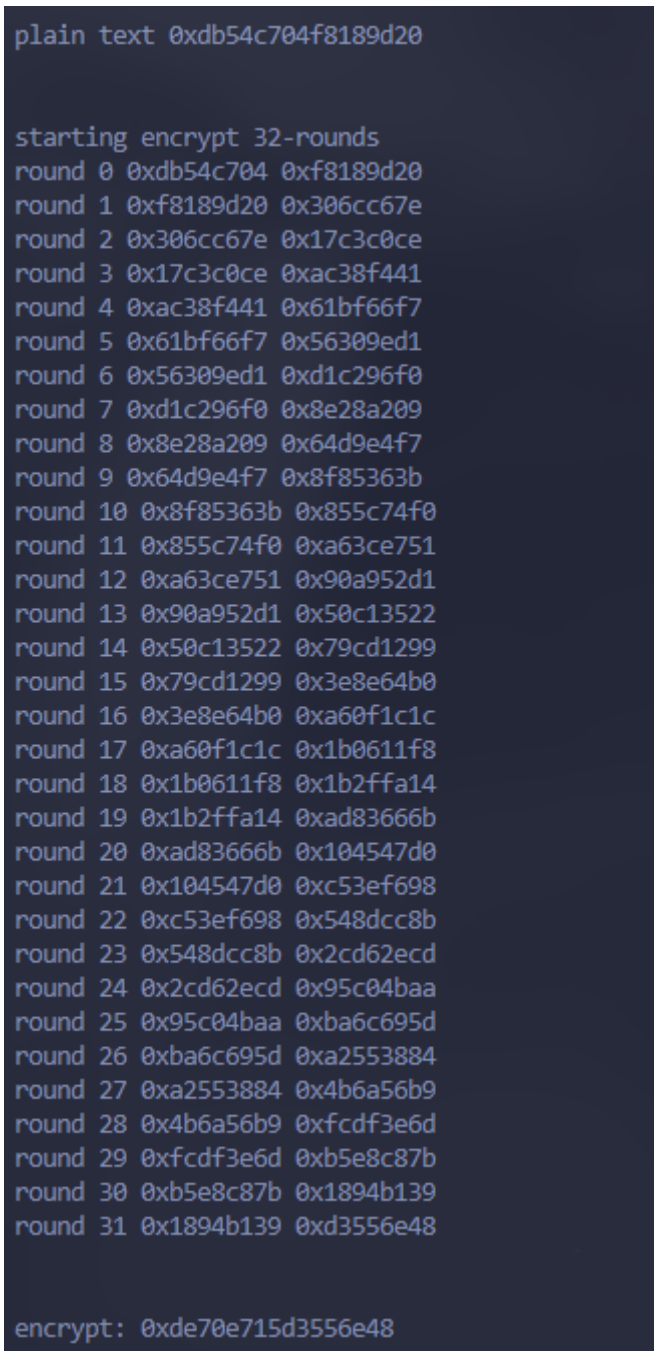
```python
# Magma (GOST) Block Cipher

pi0 = [12, 4, 6, 2, 10, 5, 11, 9, 14, 8, 13, 7, 0, 3, 15, 1]
pi1 = [6, 8, 2, 3, 9, 10, 5, 12, 1, 14, 4, 7, 11, 13, 0, 15]
pi2 = [11, 3, 5, 8, 2, 15, 10, 13, 14, 1, 7, 4, 12, 9, 6, 0]
pi3 = [12, 8, 2, 1, 13, 4, 15, 6, 7, 0, 10, 5, 3, 14, 9, 11]
pi4 = [7, 15, 5, 10, 8, 1, 6, 13, 0, 9, 3, 14, 11, 4, 2, 12]
pi5 = [5, 13, 15, 6, 9, 2, 12, 10, 11, 7, 8, 1, 4, 3, 14, 0]
pi6 = [8, 14, 2, 5, 6, 9, 1, 12, 15, 4, 11, 0, 13, 10, 3, 7]
pi7 = [1, 7, 14, 13, 0, 5, 8, 3, 4, 15, 10, 6, 9, 12, 11, 2]

pi = [pi0, pi1, pi2, pi3, pi4, pi5, pi6, pi7]

MASK32 = 2 ** 32 - 1

# вводимое число x в 32 bits
# выводимое число y в 32-bits


def t(x):
    y = 0
    for i in reversed(range(8)):
        j = (x >> 4 * i) & 0xf
        y <<= 4
        y ^= pi[i][j]
    return y


# x 32-bit integer
def rot11(x):
    return ((x << 11) ^ (x >> (32 - 11))) & MASK32
```

```python
# x и k это 32-bit integers
def g(x, k):
    return rot11(t((x + k) % 2 ** 32))


# x это 64 bits
# деление на тюплы
def split(x):
    L = x >> 32
    R = x & MASK32
    return (L, R)


# Левый и правый по 32 бита
# Возвращается 64-bits
def join(L, R):
    return (L << 32) ^ R


# k равно 256-bits.
# возвращается список из 32 ключей по 32 бита
# первые 8 ключей берутся из деления ключа на 8 частей по 32 бита
# оставшиеся ключи повотряют первые восемь
def magma_key_schedule(k):
    keys = []
    for i in reversed(range(8)):
        keys.append((k >> (32 * i)) & MASK32)
    for i in range(8):
        keys.append(keys[i])
    for i in range(8):
        keys.append(keys[i])
    for i in reversed(range(8)):
        keys.append(keys[i])
```

```
    return keys


# число x (текст) 64 bits.
# k 256 bits
# шифртекст 64-bits
def magma_encrypt(x, k):
    keys = magma_key_schedule(k)
    (L, R) = split(x)
    for i in range(31):
        print(f'round {i}', hex(L), hex(R))
        (L, R) = (R, L ^ g(R, keys[i]))

    print(f'round {i+1}', hex(L), hex(R))
    return join(L ^ g(R, keys[-1]), R)


# число x (шифртекст) is 64 bits.
# k 256 bits
# текст 64-bits
def magma_decrypt(x, k):
    keys = magma_key_schedule(k)
    keys.reverse()
    (L, R) = split(x)
    for i in range(31):
        print(f'round {i}', hex(L), hex(R))
        (L, R) = (R, L ^ g(R, keys[i]))

    print(f'round {i+1}', hex(L), hex(R))

    return join(L ^ g(R, keys[-1]), R)
```

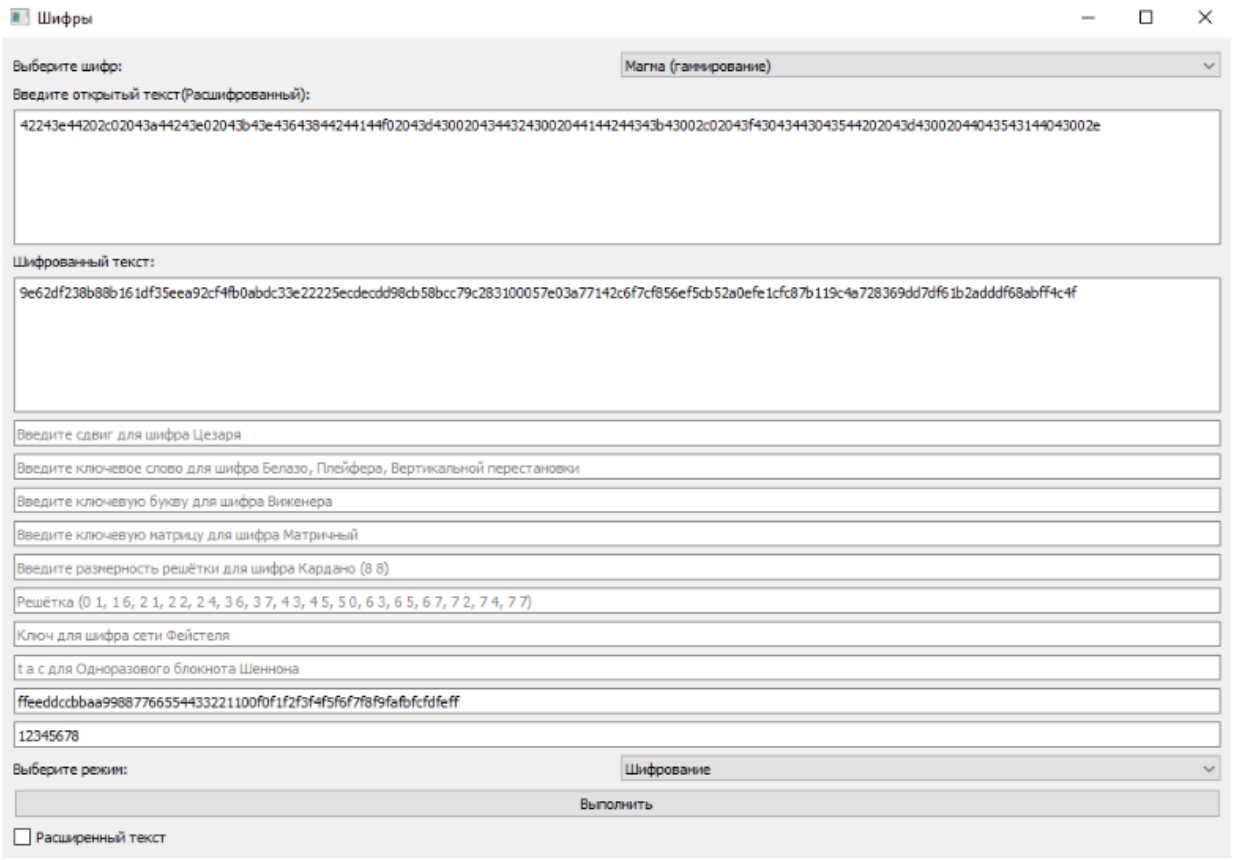
```python
def magma_cipher():
    # ключ
    k = int('ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff', 16)
    my_text = int('db54c704f8189d20', 16)   # текст
    print('\n\nplain text', hex(my_text))
    print('\n\nstarting encrypt 32-rounds')
    CT = magma_encrypt(my_text, k)   # шифртекст
    print('\n\nencrypt:', hex(CT))
    print('\n\nstarting decrypt 32-rounds')
    decrypt_text = magma_decrypt(CT, k)   # расшифровка
    print('\n\ndecrypt:', hex(decrypt_text))
    # расшифровка = текст?
    print('\n\nthey are similar?', decrypt_text == my_text)

magma_cipher()
```

Тестирование (скрин работы программы и фото карточки ручного теста с указанием ключа)



```
plain text 0xdb54c704f8189d20


starting encrypt 32-rounds
round 0 0xdb54c704 0xf8189d20
round 1 0xf8189d20 0x306cc67e
round 2 0x306cc67e 0x17c3c0ce
round 3 0x17c3c0ce 0xac38f441
round 4 0xac38f441 0x61bf66f7
round 5 0x61bf66f7 0x56309ed1
round 6 0x56309ed1 0xd1c296f0
round 7 0xd1c296f0 0x8e28a209
round 8 0x8e28a209 0x64d9e4f7
round 9 0x64d9e4f7 0x8f85363b
round 10 0x8f85363b 0x855c74f0
round 11 0x855c74f0 0xa63ce751
round 12 0xa63ce751 0x90a952d1
round 13 0x90a952d1 0x50c13522
round 14 0x50c13522 0x79cd1299
round 15 0x79cd1299 0x3e8e64b0
round 16 0x3e8e64b0 0xa60f1c1c
round 17 0xa60f1c1c 0x1b0611f8
round 18 0x1b0611f8 0x1b2ffa14
round 19 0x1b2ffa14 0xad83666b
round 20 0xad83666b 0x104547d0
round 21 0x104547d0 0xc53ef698
round 22 0xc53ef698 0x548dcc8b
round 23 0x548dcc8b 0x2cd62ecd
round 24 0x2cd62ecd 0x95c04baa
round 25 0x95c04baa 0xba6c695d
round 26 0xba6c695d 0xa2553884
round 27 0xa2553884 0x4b6a56b9
round 28 0x4b6a56b9 0xfcdf3e6d
round 29 0xfcdf3e6d 0xb5e8c87b
round 30 0xb5e8c87b 0x1894b139
round 31 0x1894b139 0xd3556e48


encrypt: 0xde70e715d3556e48
```

```
starting decrypt 32-rounds
round 0 0xde70e715 0xd3556e48
round 1 0xd3556e48 0x1894b139
round 2 0x1894b139 0xb5e8c87b
round 3 0xb5e8c87b 0xfcdf3e6d
round 4 0xfcdf3e6d 0x4b6a56b9
round 5 0x4b6a56b9 0xa2553884
round 6 0xa2553884 0xba6c695d
round 7 0xba6c695d 0x95c04baa
round 8 0x95c04baa 0x2cd62ecd
round 9 0x2cd62ecd 0x548dcc8b
round 10 0x548dcc8b 0xc53ef698
round 11 0xc53ef698 0x104547d0
round 12 0x104547d0 0xad83666b
round 13 0xad83666b 0x1b2ffa14
round 14 0x1b2ffa14 0x1b0611f8
round 15 0x1b0611f8 0xa60f1c1c
round 16 0xa60f1c1c 0x3e8e64b0
round 17 0x3e8e64b0 0x79cd1299
round 18 0x79cd1299 0x50c13522
round 19 0x50c13522 0x90a952d1
round 20 0x90a952d1 0xa63ce751
round 21 0xa63ce751 0x855c74f0
round 22 0x855c74f0 0x8f85363b
round 23 0x8f85363b 0x64d9e4f7
round 24 0x64d9e4f7 0x8e28a209
round 25 0x8e28a209 0xd1c296f0
round 26 0xd1c296f0 0x56309ed1
round 27 0x56309ed1 0x61bf66f7
round 28 0x61bf66f7 0xac38f441
round 29 0xac38f441 0x17c3c0ce
round 30 0x17c3c0ce 0x306cc67e
round 31 0x306cc67e 0xf8189d20


decrypt: 0xdb54c704f8189d20
```



Ключ

K = ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff.

A.2.2 Режим гаммирования

A.2.2.1 Зашифрование

$s = n = 64$,

IV = 12345678.

Т а б л и ц а А.8 – Зашифрование в режиме гаммирования

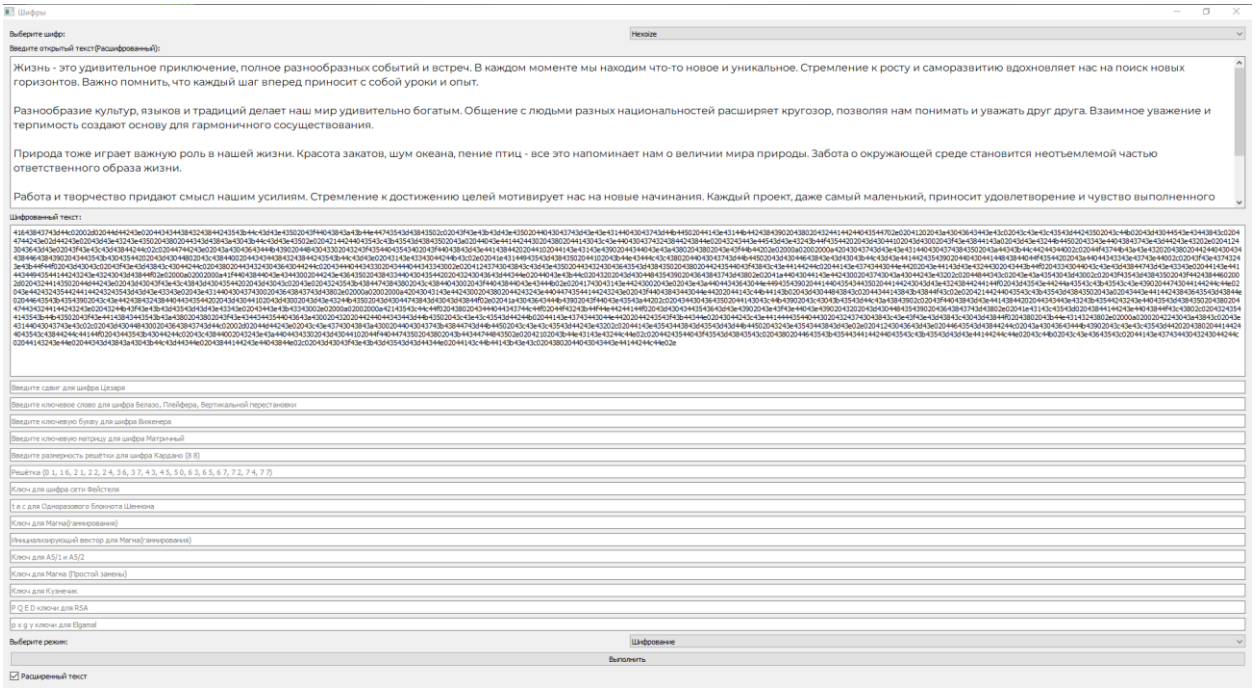
i	1	2
P_i	92def06b3c130a59	db54c704f8189d20
Входной блок	1234567800000000	1234567800000001
Выходной блок	dc46e167aba4b365	e571ca972ef0c049
C_i	4e98110c 97b7b93c	3e250d93d6e85d69

окончание таблицы А.8

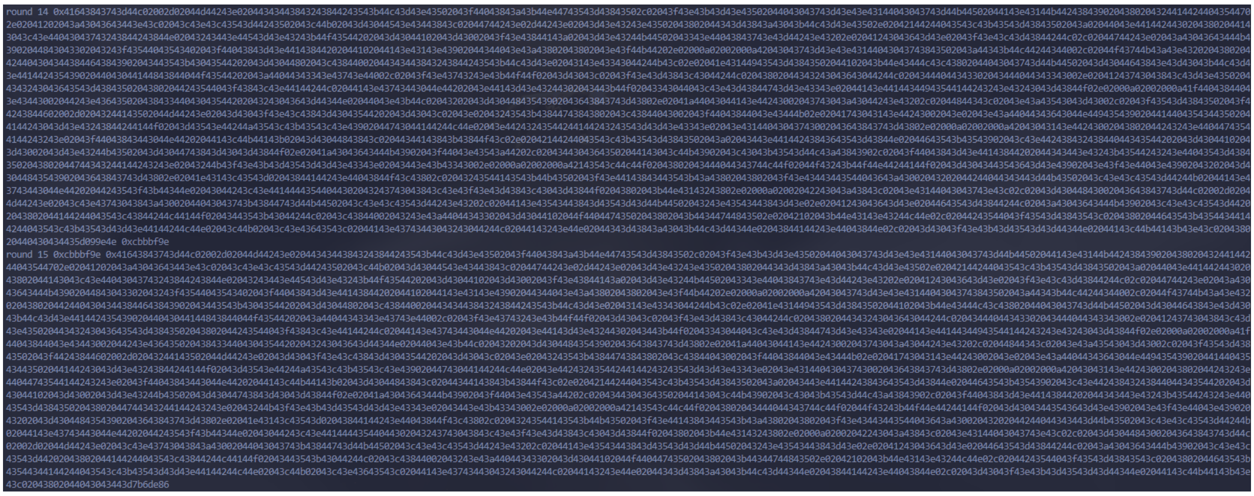
i	3	4
P_i	4a98fb2e67a8024c	8912409b17b57e41
Входной блок	1234567800000002	1234567800000003
Выходной блок	59f57da6601ad9a3	df9cf61bbce7df6c
C_i	136d868807b2dbef	568eb680ab52a12d

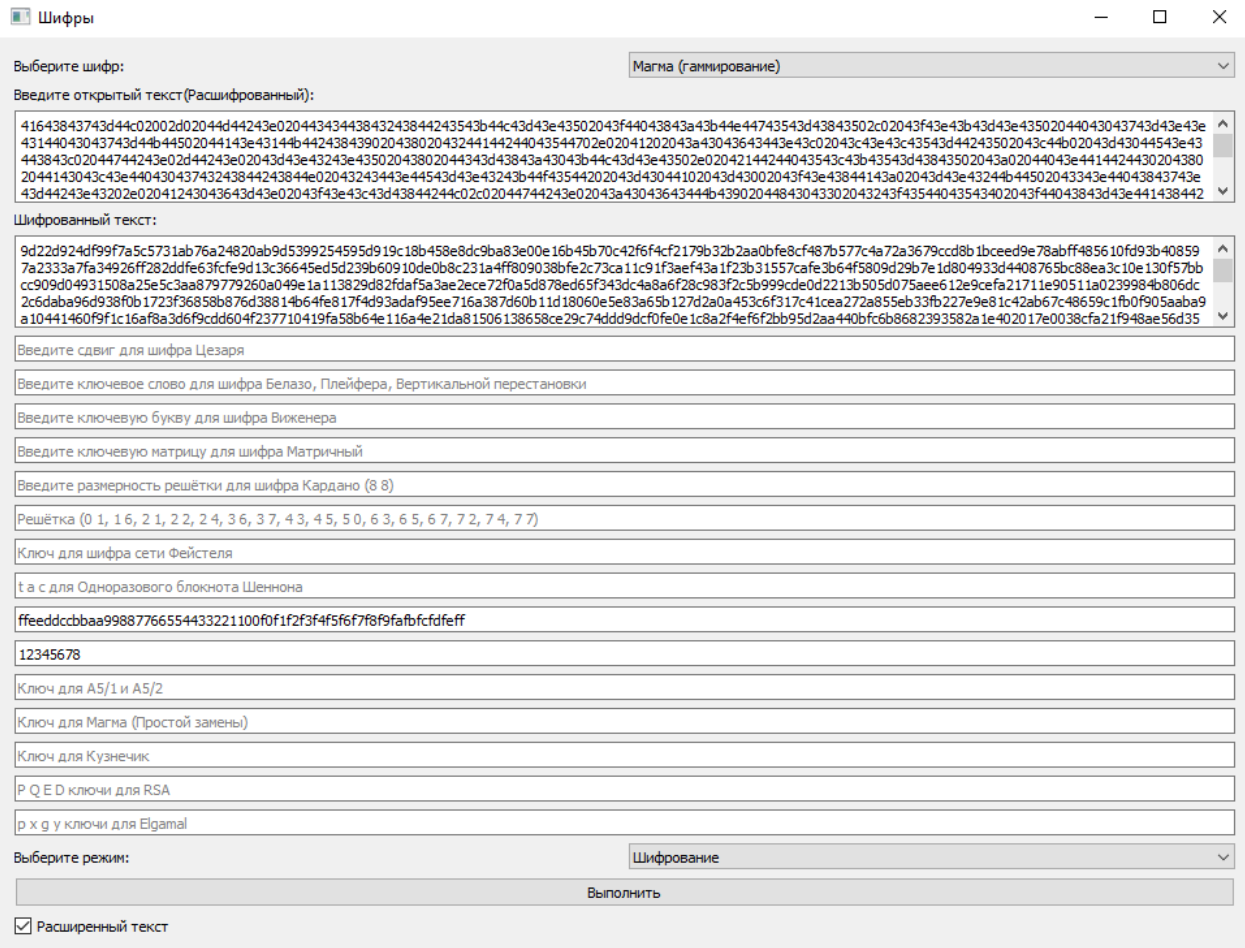
Работа с текстом не менее 1000 знаков (шифрование и расшифрование с указанием ключа)

Текст в hex

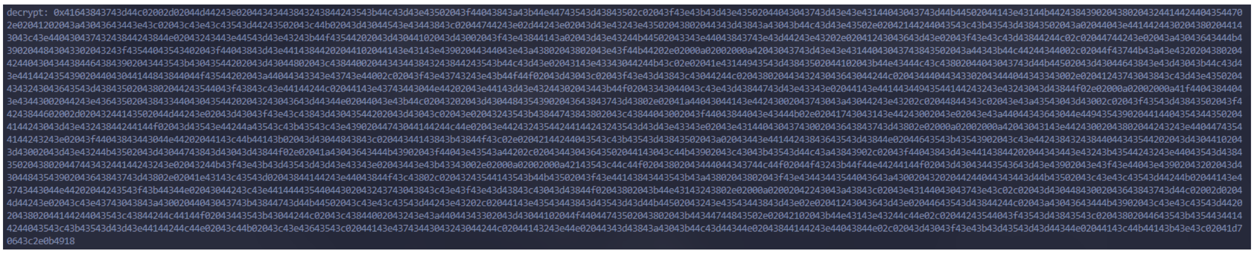


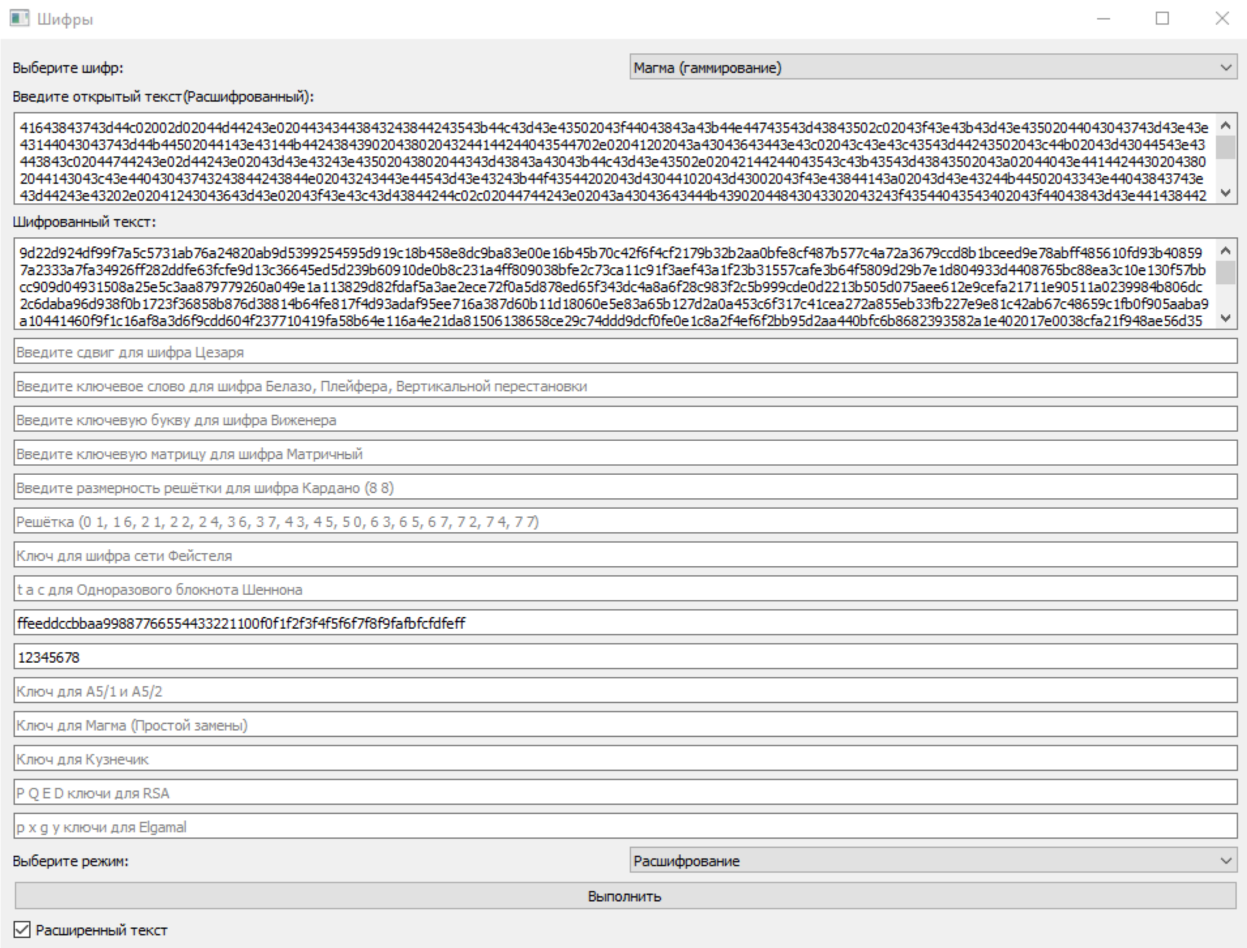
Зашифрование



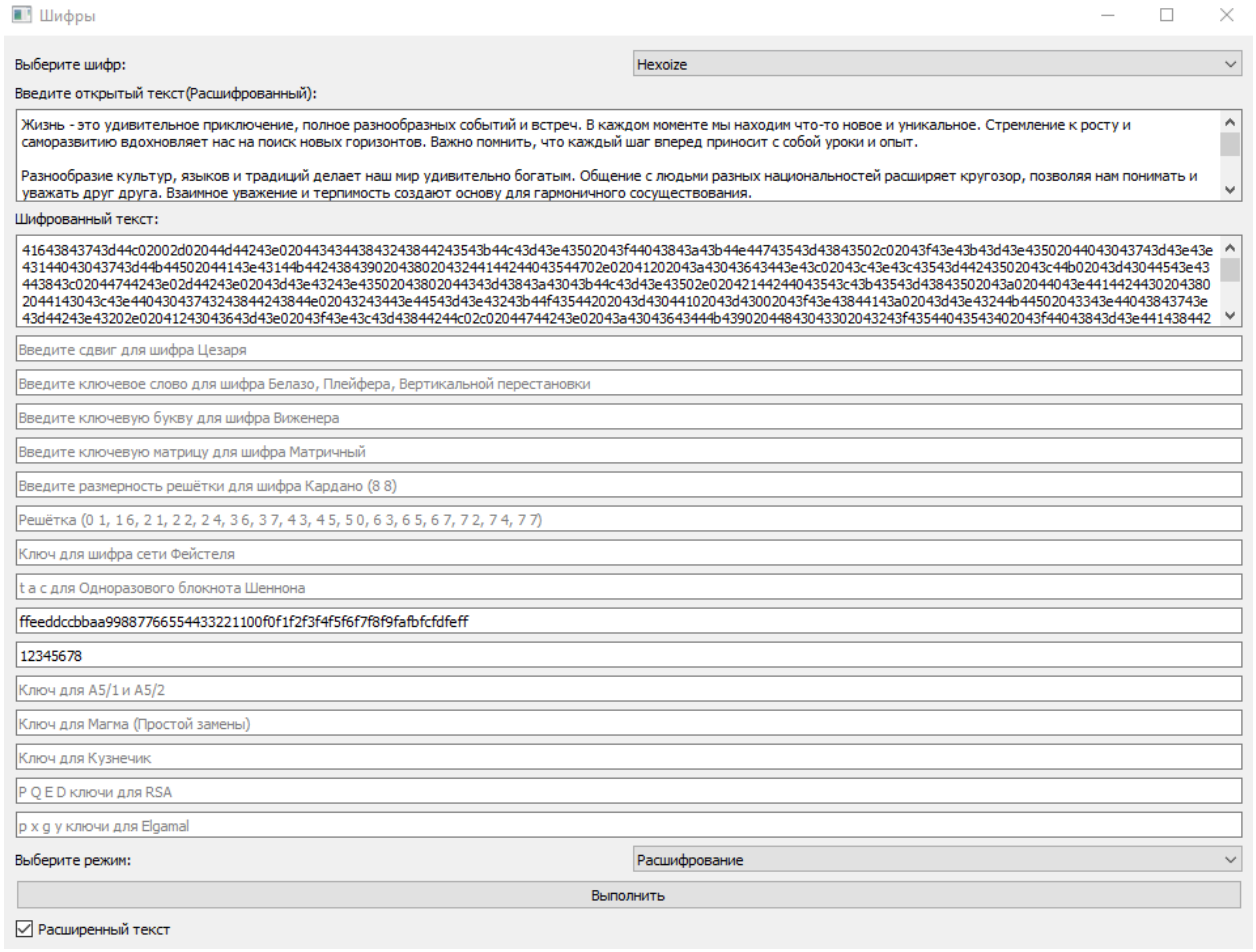


Расшифрование





Hex в текст


Шифры
Выберите шифр:
Hexoize
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.
Шифрованный текст:
41643843743d44c02002d02044d44243e020443434438432438442435 43b44c43d43e43502043f44043843a43b44e44743543d43843502c02043f43e43b43d43e435020440430437 43d43e43e43144043043743d44b44502044143e43144b44243843902043802043244144244043544702e02041202043a43043643443e43c02043c43e43c43543d44243502043c44b02043d43044543e43443843c02044744243e02d44243e02043d43e43243e435020438020443 43d43843a43043b44c43d43e43502e020421442440435 43c43b43543d43843502043a02044043e44144244302043802044143043c43e44043043743243844243844e02043243443e44543d43e43243b44f43544202043d43044102043d43002043f43e43844143a02043d43e43244b44502043343e44043843743e43d44243e43202e020412430436 43d43e02043f43e43c43d43844244c02c02044744243e02043a43043643444b43902044843043302043243f43544043543402043f44043843d43e441438442
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
t a c для Одноразового блокнота Шеннона
ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
12345678
Ключ для A5/1 и A5/2
Ключ для Магма (Простой замены)
Ключ для Кузнечик
P Q E D ключи для RSA
p x g y ключи для Elgamal
Выберите режим:
Расшифрование
Выполнить
Расширенный текст

20.КУЗНЕЧИК

Шифр Кузнечик - симметричный блочный шифр, основан на принципе замены и перестановки битов входного текста с использованием ключа. Процесс шифрования включает в себя множество раундов замены и перестановки. Размер блока в шифре Кузнечик составляет 128 бит, ключа 256 бит.

Блок-схема программы

Код программы с комментариями

```
pi = [252, 238, 221, 17, 207, 110, 49, 22, 251, 196, 250, 218, 35, 197, 4, 77,
      233, 119, 240, 219, 147, 46, 153, 186, 23, 54, 241, 187, 20, 205, 95, 193,
      249, 24, 101, 90, 226, 92, 239, 33, 129, 28, 60, 66, 139, 1, 142, 79,
      5, 132, 2, 174, 227, 106, 143, 160, 6, 11, 237, 152, 127, 212, 211, 31,
      235, 52, 44, 81, 234, 200, 72, 171, 242, 42, 104, 162, 253, 58, 206, 204,
      181, 112, 14, 86, 8, 12, 118, 18, 191, 114, 19, 71, 156, 183, 93, 135,
      21, 161, 150, 41, 16, 123, 154, 199, 243, 145, 120, 111, 157, 158, 178, 177,
      50, 117, 25, 61, 255, 53, 138, 126, 109, 84, 198, 128, 195, 189, 13, 87,
      223, 245, 36, 169, 62, 168, 67, 201, 215, 121, 214, 246, 124, 34, 185, 3,
      224, 15, 236, 222, 122, 148, 176, 188, 220, 232, 40, 80, 78, 51, 10, 74,
      167, 151, 96, 115, 30, 0, 98, 68, 26, 184, 56, 130, 100, 159, 38, 65,
      173, 69, 70, 146, 39, 94, 85, 47, 140, 163, 165, 125, 105, 213, 149, 59,
      7, 88, 179, 64, 134, 172, 29, 247, 48, 55, 107, 228, 136, 217, 231, 137,
      225, 27, 131, 73, 76, 63, 248, 254, 141, 83, 170, 144, 202, 216, 133, 97,
      32, 113, 103, 164, 45, 43, 9, 91, 203, 155, 37, 208, 190, 229, 108, 82,
      89, 166, 116, 210, 230, 244, 180, 192, 209, 102, 175, 194, 57, 75, 99, 182]

pi_inv = [165, 45, 50, 143, 14, 48, 56, 192, 84, 230, 158, 57, 85, 126, 82, 145,
          100, 3, 87, 90, 28, 96, 7, 24, 33, 114, 168, 209, 41, 198, 164, 63,
          224, 39, 141, 12, 130, 234, 174, 180, 154, 99, 73, 229, 66, 228, 21, 183,
          200, 6, 112, 157, 65, 117, 25, 201, 170, 252, 77, 191, 42, 115, 132, 213,
          195, 175, 43, 134, 167, 177, 178, 91, 70, 211, 159, 253, 212, 15, 156, 47,
          155, 67, 239, 217, 121, 182, 83, 127, 193, 240, 35, 231, 37, 94, 181, 30,
          162, 223, 166, 254, 172, 34, 249, 226, 74, 188, 53, 202, 238, 120, 5, 107,
          81, 225, 89, 163, 242, 113, 86, 17, 106, 137, 148, 101, 140, 187, 119, 60,
          123, 40, 171, 210, 49, 222, 196, 95, 204, 207, 118, 44, 184, 216, 46, 54,
          219, 105, 179, 20, 149, 190, 98, 161, 59, 22, 102, 233, 92, 108, 109, 173,
          55, 97, 75, 185, 227, 186, 241, 160, 133, 131, 218, 71, 197, 176, 51, 250,
          150, 111, 110, 194, 246, 80, 255, 93, 169, 142, 23, 27, 151, 125, 236, 88,
          247, 31, 251, 124, 9, 13, 122, 103, 69, 135, 220, 232, 79, 29, 78, 4,
          235, 248, 243, 62, 61, 189, 138, 136, 221, 205, 11, 19, 152, 2, 147, 128,
```

```
          144, 208, 36, 52, 203, 237, 244, 206, 153, 16, 68, 64, 146, 58, 1, 38,
          18, 26, 72, 104, 245, 129, 139, 199, 214, 32, 10, 8, 0, 76, 215, 116]


# На вход и выход подаются 128 битные блоки
# Реализация функция преобразования S
def S(x):
    y = 0
    for i in reversed(range(16)):
        y <<= 8
        y ^= pi[(x >> (8 * i)) & 0xff]
    return y


# На вход и выход подаются 128 битные блоки
# Реализация обратной функции преобразования S
def S_inv(x):
    y = 0
    for i in reversed(range(16)):
        y <<= 8
        y ^= pi_inv[(x >> (8 * i)) & 0xff]
    return y


# На вход подаются неотрицательные целые числа
# Связанные с ними двоичные многочлены умножаются
# Возвращается целое число
def multiply_ints_as_polynomials(x, y):
    if x == 0 or y == 0:
        return 0
    z = 0
    while x != 0:
        if x & 1 == 1:
```

```
            z ^= y
        y <<= 1
        x >>= 1
    return z


# Returns the number of bits that are used
# to store the positive integer integer x.
def number_bits(x):
    nb = 0
    while x != 0:
        nb += 1
        x >>= 1
    return nb


# x is a nonnegative integer
# m is a positive integer
def mod_int_as_polynomial(x, m):
    nbm = number_bits(m)
    while True:
        nbx = number_bits(x)
        if nbx < nbm:
            return x
        mshift = m << (nbx - nbm)
        x ^= mshift


# x,y are 8-bits
# The output value is 8-bits
def kuznyechik_multiplication(x, y):
    z = multiply_ints_as_polynomials(x, y)
    m = int('111000011', 2)
```

```
    return mod_int_as_polynomial(z, m)




# The input x is 128-bits (considered as a vector of sixteen bytes)
# The return value is 8-bits
def kuznyechik_linear_functional(x):
    C = [148, 32, 133, 16, 194, 192, 1, 251, 1, 192, 194, 16, 133, 32, 148, 1]
    y = 0
    while x != 0:
        y ^= kuznyechik_multiplication(x & 0xff, C.pop())
        x >>= 8
    return y


# На вход и выход подаются 128 битные блоки
# Реализация функции R
def R(x):
    a = kuznyechik_linear_functional(x)
    return (a << 8 * 15) ^ (x >> 8)


# На вход и выход подаются 128 битные блоки
# Реализация обраной функции R
def R_inv(x):
    a = x >> 15 * 8
    x = (x << 8) & (2 ** 128 - 1)
    b = kuznyechik_linear_functional(x ^ a)
    return x ^ b


# На вход и выход подаются 128 битные блоки
```

```python
# Реализация функции L
def L(x):
    for _ in range(16):
        x = R(x)
    return x


# На вход и выход подаются 128 битные блоки
# Реализация обратной функции L
def L_inv(x):
    for _ in range(16):
        x = R_inv(x)
    return x


# k - ключ, является 256-bits
# Алгоритм развертывания ключей
def kuznyechik_key_schedule(k):
    keys = []
    a = k >> 128
    b = k & (2 ** 128 - 1)
    keys.append(a)
    keys.append(b)
    for i in range(4):
        for j in range(8):
            c = L(8 * i + j + 1)
            (a, b) = (L(S(a ^ c)) ^ b, a)
        keys.append(a)
        keys.append(b)
    return keys


# Входной текст 128 бит
```

```python
# Ключ 256 бит
def kuznyechik_encrypt(x, k):
    keys = kuznyechik_key_schedule(k)
    # запуск раундов
    for round in range(9):
        print(hex(x))
        x = L(S(x ^ keys[round]))

    print(hex(x))
    return x ^ keys[-1]


# The ciphertext x is 128-bits
# The key k is 256-bits
def kuznyechik_decrypt(x, k):
    keys = kuznyechik_key_schedule(k)
    keys.reverse()
    for round in range(9):
        print(hex(x))
        x = S_inv(L_inv(x ^ keys[round]))
    print(hex(x))
    return x ^ keys[-1]


def kuznyechik_cipher():
    # the key
    k = int('8899aabbccddeeff0011223344556677fedcba98765432100123456789abcdef', 16)
    my_text =
int('0x42243e44202c02043a44243e02043b43e43643844244144f02043d43002043443243002044144244343b43002c02043f4304344304354cb5b7b072d
f04ca7', 16)
    print('\n\nplain text', hex(my_text))
    print('\n\nstarting encrypt 9-rounds')
    CT = kuznyechik_encrypt(my_text, k)
```

```
    print('\n\nencrypt:', hex(CT))
    print('\n\nstarting decrypt 9-rounds')
    decrypt_text = kuznyechik_decrypt(CT, k)
    print('\n\ndecrypt:', hex(decrypt_text))
    # Сравнение текста исходного и конечного
    print('\n\nthey are similar?', decrypt_text == my_text)

kuznyechik_cipher()
```

Тестирование (скрин работы программы и фото карточки ручного теста с указанием ключа)

А.1 Блочный шифр с длиной блока *n* = 128 бит

Примеры используют следующие параметры:

Ключ

K = 8899aabbccddeeff0011223344556677fedcba98765432100123456789abcdef.

Открытый текст – четыре 128-битных блока:

P_1 = 1122334455667700ffeeddccbbaa9988,
P_2 = 00112233445566778899aabbcceeff0a,
P_3 = 11223344556677889 9aabbcceeff0a00,
P_4 = 2233445566778899aabbcceeff0a0011.

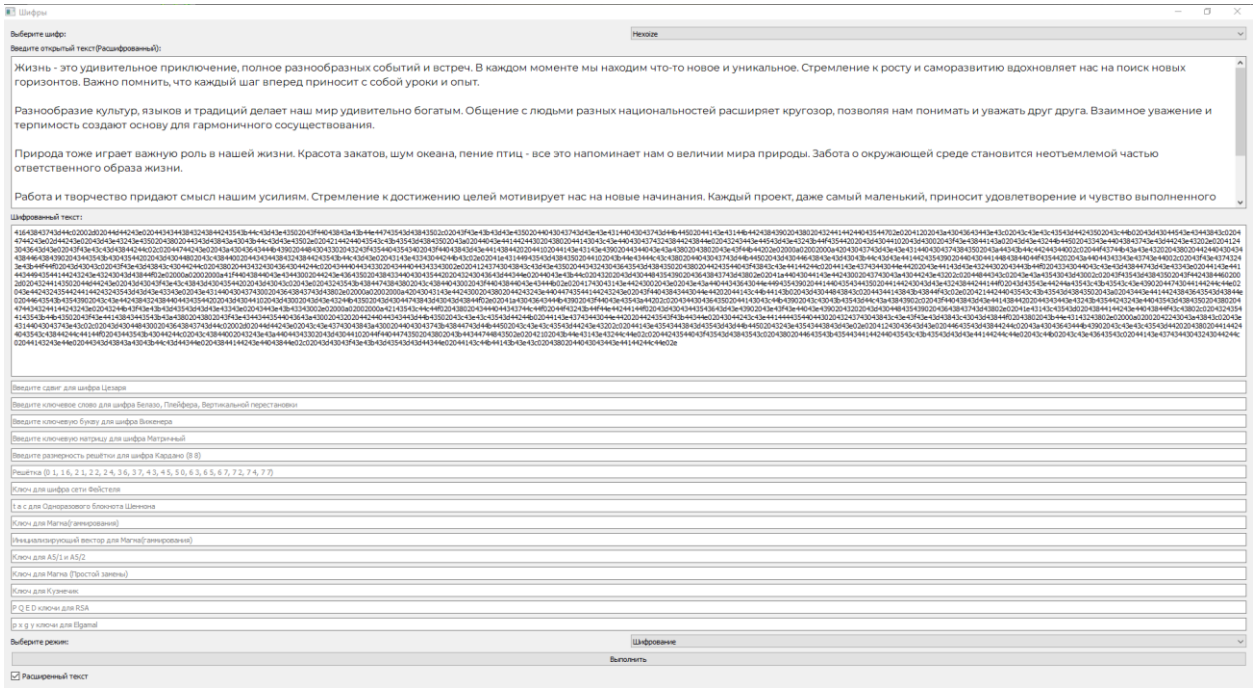
А.1.1 Режим простой замены

Таблица А.1 – Зашифрование в режиме простой замены

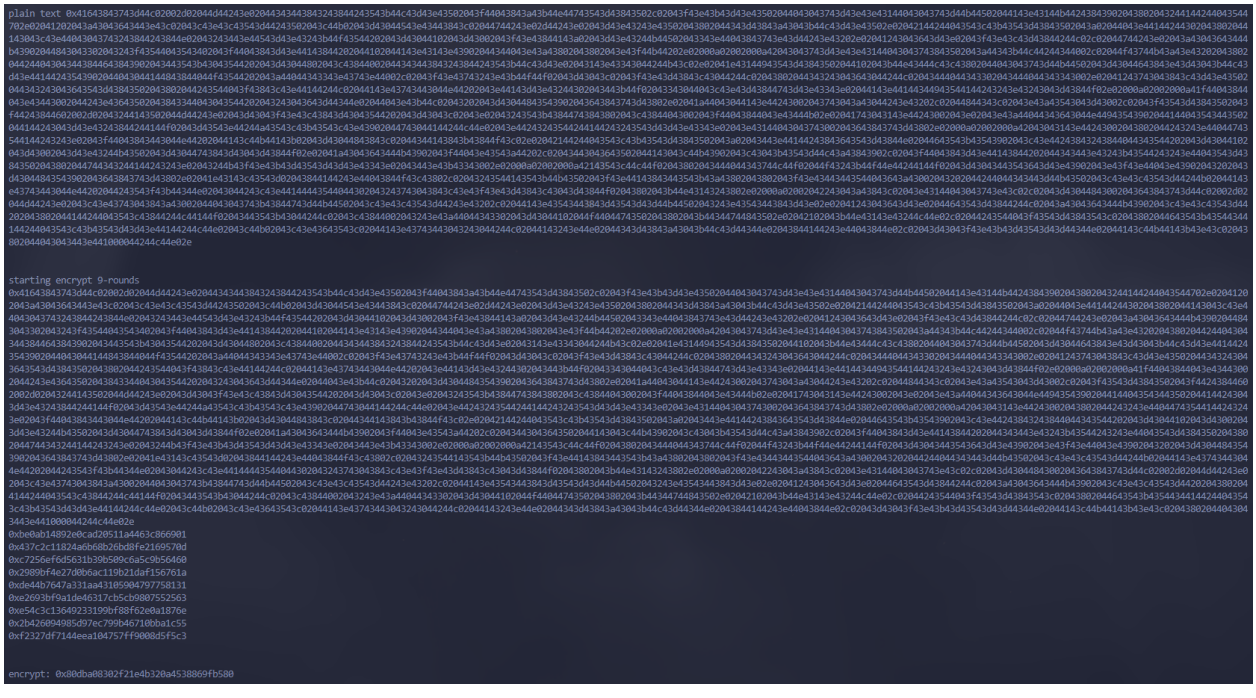
Открытый текст	Шифртекст
1122334455667700ffeeddccbbaa9988	7f679d90bebc24305a468d42b9d4edcd
00112233445566778899aabbcceeff0a	b429912c6e0032f9285452d76718d08b
11223344556677889 9aabbcceeff0a00	f0ca33549d247ceef3f5a5313bd4b157
2233445566778899aabbcceeff0a0011	d0b09ccde830b9eb3a02c4c5aa8ada98

Работа с текстом не менее 1000 знаков (шифрование и расшифрование с указанием ключа)

Текст в hex



Зашифрование



Расшифрование

```
starting decrypt 9-rounds
0x80dba08302f21e4b320a4538869fb580
0x900682c7e09aa6cfdc686902c60cf942
0xbf3519121f0dee274cd21ec0c2277eea
0xb38f7b8cdf5903ef0ce2ecbe5d55bdd2
0x634323516c6f7e968437786fa3afd92a
0x7eeddb26e39ae884c27eb39c057f87b6
0xfa603d2e3cacf751a0e26b61c32a9964
0x984d64429723282b90e6b211adaedb49
0x40d60bd0e4b4f8c20432e121b52da4ee
0x8cd9e9bff89e0abe0011661708118659


decrypt: 0x44043043443e441000044244c44e02e
```

Hex в текст