Кузнечик

```
import pickle
import binascii

text = input("Введите текст поговорки: ")
count = 0
while len(text) % 16 != 0:
    text += "Я"
    count += 1


### Кузнечик ###
class kuznechik:
    def __init__(self, key):
        self.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)
        self.piinv = (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)
        # Предварительно вычисленная таблица с результатами умножения в поле x^8
+ x^7 + x^6 + x + 1
        f = open('C:\\Users\\Sergey\\Downloads\\gost_tables', 'rb')
        self.multtable = pickle.load(f)
        f.close()
        # Константы C, использующиеся для ключевого расписания
        self.C = [[110, 162, 118, 114, 108, 72, 122, 184, 93, 39, 189, 16, 221,
132, 148, 1],
                  [220, 135, 236, 228, 216, 144, 244, 179, 186, 78, 185, 32, 121,
203, 235, 2],
                  [178, 37, 154, 150, 180, 216, 142, 11, 231, 105, 4, 48, 164,
79, 127, 3],
                  [123, 205, 27, 11, 115, 227, 43, 165, 183, 156, 177, 64, 242,
85, 21, 4],
                  [21, 111, 109, 121, 31, 171, 81, 29, 234, 187, 12, 80, 47, 209,
129, 5],
                  [167, 74, 247, 239, 171, 115, 223, 22, 13, 210, 8, 96, 139,
158, 254, 6],
                  [201, 232, 129, 157, 199, 59, 165, 174, 80, 245, 181, 112, 86,
26, 106, 7],
                  [246, 89, 54, 22, 230, 5, 86, 137, 173, 251, 161, 128, 39, 170,
42, 8],
                  [152, 251, 64, 100, 138, 77, 44, 49, 240, 220, 28, 144, 250,
46, 190, 9],
                  [42, 222, 218, 242, 62, 149, 162, 58, 23, 181, 24, 160, 94, 97,
193, 10],
                  [68, 124, 172, 128, 82, 221, 216, 130, 74, 146, 165, 176, 131,
229, 85, 11],
                  [141, 148, 45, 29, 149, 230, 125, 44, 26, 103, 16, 192, 213,
255, 63, 12],
                  [227, 54, 91, 111, 249, 174, 7, 148, 71, 64, 173, 208, 8, 123,
171, 13],
                  [81, 19, 193, 249, 77, 118, 137, 159, 160, 41, 169, 224, 172,
52, 212, 14],
                  [63, 177, 183, 139, 33, 62, 243, 39, 253, 14, 20, 240, 113,
176, 64, 15],
                  [47, 178, 108, 44, 15, 10, 172, 209, 153, 53, 129, 195, 78,
151, 84, 16],
```

```
                [65, 16, 26, 94, 99, 66, 214, 105, 196, 18, 60, 211, 147, 19,
192, 17],
                [243, 53, 128, 200, 215, 154, 88, 98, 35, 123, 56, 227, 55, 92,
191, 18],
                [157, 151, 246, 186, 187, 210, 34, 218, 126, 92, 133, 243, 234,
216, 43, 19],
                [84, 127, 119, 39, 124, 233, 135, 116, 46, 169, 48, 131, 188,
194, 65, 20],
                [58, 221, 1, 85, 16, 161, 253, 204, 115, 142, 141, 147, 97, 70,
213, 21],
                [136, 248, 155, 195, 164, 121, 115, 199, 148, 231, 137, 163,
197, 9, 170, 22],
                [230, 90, 237, 177, 200, 49, 9, 127, 201, 192, 52, 179, 24,
141, 62, 23],
                [217, 235, 90, 58, 233, 15, 250, 88, 52, 206, 32, 67, 105, 61,
126, 24],
                [183, 73, 44, 72, 133, 71, 128, 224, 105, 233, 157, 83, 180,
185, 234, 25],
                [5, 108, 182, 222, 49, 159, 14, 235, 142, 128, 153, 99, 16,
246, 149, 26],
                [107, 206, 192, 172, 93, 215, 116, 83, 211, 167, 36, 115, 205,
114, 1, 27],
                [162, 38, 65, 49, 154, 236, 209, 253, 131, 82, 145, 3, 155,
104, 107, 28],
                [204, 132, 55, 67, 246, 164, 171, 69, 222, 117, 44, 19, 70,
236, 255, 29],
                [126, 161, 173, 213, 66, 124, 37, 78, 57, 28, 40, 35, 226, 163,
128, 30],
                [16, 3, 219, 167, 46, 52, 95, 246, 100, 59, 149, 51, 63, 39,
20, 31],
                [94, 167, 216, 88, 30, 20, 155, 97, 241, 106, 193, 69, 156,
237, 168, 32]]
        self.roundkey = [key[:16], key[16:]]
        self.roundkey = self.roundkey + self.keyschedule(self.roundkey)  #
Просчёт раундовых ключей

    # Дополнение в поле x^8 + x^7 + x^6 + x + 1
    def add_field(self, x, y):
        return x ^ y

    # Суммирование всех x элементов
    def sum_field(self, x):
        s = 0
        for a in x:
            s ^= a
        return s

    # Умножение в поле
    def mult_field(self, x, y):
        p = 0
        while x:
```

```
            if x & 1:
                p ^= y
            if y & 0x80:
                y = (y << 1) ^ 0x1C3
            else:
                y <<= 1
            x >>= 1
        return p

    # XOR двоичных строк x и k
    def xtransformation(self, x, k):
        return [x[i] ^ k[i] for i in range(len(k))]

    # возвращает элемент по счёту x из pi таблицы
    def pitransformation(self, x):
        return self.pi[x]

    ## Обратное преобразование pi
    def piinvtransformation(self, x):
        return self.piinv[x]

    # Замена каждого байта в x соответствующим байтом из таблицы pi
    def stransformation(self, x):
        return [self.pitransformation(x[i]) for i in range(len(x))]

    ## Обратное преобразование s
    def sinvtransformation(self, x):
        return [self.piinvtransformation(i) for i in x]

    # Преобразование списка байтов в однобайтовый с использованием арифметики с
конечным полем
    def l(self, x):
        consts = [148, 32, 133, 16, 194, 192, 1, 251, 1, 192, 194, 16, 133, 32,
148, 1]
        multiplication = [self.multtable[x[i]][consts[i]] for i in range(len(x))]
        return self.sum_field(multiplication)

    # R преобразование добавить список байтов с результатом l-преобразования
    def rtransformation(self, x):
        return [self.l(x), ] + x[:-1]

    ## Обратное преобразование R
    def rinvtransformation(self, x):
        return x[1:] + [self.l(x[1:] + [x[0], ]), ]

    # Преобразование L по стандарту
    def ltransformation(self, x):
        for i in range(len(x)):
            x = self.rtransformation(x)
        return x
```

```
    ## Обратное преобразование L
    def linvtransformation(self, x):
        for i in range(len(x)):
            x = self.rinvtransformation(x)
        return x

    # преобразование F с использованием ключевого расписания
    def ftransformation(self, k, a):
        tmp = self.xtransformation(k, a[0])
        tmp = self.stransformation(tmp)
        tmp = self.ltransformation(tmp)
        tmp = self.xtransformation(tmp, a[1])
        return [tmp, a[0]]

    # Создание раундового ключа
    def keyschedule(self, roundkey):
        roundkeys = []
        for i in range(4):
            for k in range(8):
                roundkey = self.ftransformation(self.C[8 * i + k], roundkey)
            roundkeys.append(roundkey[0])
            roundkeys.append(roundkey[1])
        return roundkeys

    def encryption(self, m):
        for i in range(9):
            m = self.xtransformation(m, self.roundkey[i])  # Выполнение XOR
текущего значения байт с итерационым ключом
            m = self.stransformation(m)  # Замена байт текущего значения массива
по таблице замен
            m = self.ltransformation(m)  # произведение в поле Галуа
        m = self.xtransformation(m, self.roundkey[9])  # Выполнение XOR
        return m

    def decryption(self, c):
        for i in range(9, 0, -1):
            c = self.xtransformation(c, self.roundkey[i])
            c = self.linvtransformation(c)
            c = self.sinvtransformation(c)
        c = self.xtransformation(c, self.roundkey[0])
        return c


print("Проверка контрольными примерами:")
k = '8899aabbccddeeff0011223344556677fedcba98765432100123456789abcdef'
mtest = list(binascii.unhexlify('00112233445566778899aabbcceeff0a'))
ktest =
list(binascii.unhexlify('8899aabbccddeeff0011223344556677fedcba98765432100123456789abcdef'))

gost = kuznechik(ktest)
```

```
c = gost.encryption(mtest)
d = gost.decryption(c)

test1 = binascii.hexlify(bytearray(c)), b'b429912c6e0032f9285452d76718d08b'
test2 = binascii.hexlify(bytearray(d)), b'00112233445566778899aabbcceeff0a'

print("Шифрование: 00112233445566778899aabbcceeff0a")
print("Используемый ключ: ", k)
print("Результат // Эталон")
print(test1)
print("Расшифрование: b429912c6e0032f9285452d76718d08b")
print("Результат  // Эталон")
print(test2)


def ghopper(text):
    temp1, temp2 = [], []

    b = binascii.hexlify(bytes(str.encode(text)))
    test = list(binascii.unhexlify(b))

    for i in range(0, len(test), 16):
        c = gost.encryption(test)
        test = test[16:]
        temp1 += c

    print(" ")
    print("Кузнечик")
    print(binascii.hexlify(bytearray(temp1)))
    print("Ключ:", k)

    for i in range(0, len(temp1), 16):
        d = gost.decryption(temp1)
        d = binascii.hexlify(bytearray(d))  # добавление защифрованного блока к
массиву
        d = binascii.unhexlify(d).decode('utf-8')  # декодирование в utf-8
        temp1 = temp1[16:]
        temp2 += d

    decrypt = ""
    decrypt += ''.join(str(e) for e in temp2)
    print(decrypt[:-count])
    return decrypt[:-count]


ghopper(text)
```

Пример работы:

```
PS C:\Users\Sergey\Desktop\УЧЁБА\Крипта\2sem> & C:/Users/Sergey/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Sergey/Desktop/УЧЁБА/Крипта/2sem/Block_G/kuznechik.py
Введите текст: неттакогомудрецазптвкоторомнебылобыглупоститчк
Проверка контрольными примерами:
Шифрование: 00112233445566778899aabbcceeff0a
Используемый ключ:  8899aabbccddeeff0011223344556677fedcba98765432100123456789abcdef
Результат // Эталон
(b'b429912c6e0032f9285452d76718d08b', b'b429912c6e0032f9285452d76718d08b')
Расшифрование: b429912c6e0032f9285452d76718d08b
Результат  // Эталон
(b'00112233445566778899aabbcceeff0a', b'00112233445566778899aabbcceeff0a')

Кузнечик
b'caace248945dc3a4420f1ca2ad8ebe2221c6ae49400d7ead64506023c32c5dc287304c55750f1faf265eccf20415d362996abf9f091e717ccad6496d0de9043446574cae40d437ca12db1e1da21e8c03e5008085b5c6ffc68654768e8cebd4fa'
Ключ: 8899aabbccddeeff0011223344556677fedcba98765432100123456789abcdef
неттакогомудрецазптвкоторомнебылобыглупоститчк
PS C:\Users\Sergey\Desktop\УЧЁБА\Крипта\2sem>
```

Магма

```python
def GOST_Magma_Add_32(a, b, c):
    internal = 0
    for i in range(3, -1, -1):
        internal = a[i] + b[i] + (internal >> 8)
        c[i] = internal & 0xff

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

def GOST_Magma_T(in_data, out_data):
    for i in range(4):
        first_part_byte = (in_data[i] & 0xf0) >> 4
        sec_part_byte = (in_data[i] & 0x0f)
        first_part_byte = Pi[i * 2][first_part_byte]
        sec_part_byte = Pi[i * 2 + 1][sec_part_byte]
        out_data[i] = (first_part_byte << 4) | sec_part_byte

def GOST_Magma_g(k, a, out_data):
    internal = [0] * 4
    out_data_32 = 0

    GOST_Magma_Add_32(a, k, internal)
    GOST_Magma_T(internal, internal)

    out_data_32 = (internal[0] << 24) + (internal[1] << 16) + (internal[2] << 8)
+ internal[3]
    out_data_32 = ((out_data_32 << 11) | (out_data_32 >> 21)) & 0xFFFFFFFF

    out_data[3] = out_data_32 & 0xFF
    out_data[2] = (out_data_32 >> 8) & 0xFF
    out_data[1] = (out_data_32 >> 16) & 0xFF
    out_data[0] = (out_data_32 >> 24) & 0xFF

def GOST_Magma_Add(a, b, c):
```

```
    for i in range(len(a)):
        c[i] = a[i] ^ b[i]


def GOST_Magma_G(k, a, out_data):
    a_0 = [0] * 4  # Правая часть блока
    a_1 = [0] * 4  # Левая часть блока
    G = [0] * 4

    # Разделить 64-битный входной блок на две части
    for i in range(4):
        a_0[i] = a[4 + i]
        a_1[i] = a[i]

    # Применить преобразование g
    GOST_Magma_g(k, a_0, G)

    # Применить XOR результата преобразования g к левой части блока
    GOST_Magma_Add(a_1, G, G)

    for i in range(4):
        # Скопировать значение из правой части в левую часть
        a_1[i] = a_0[i]
        # Скопировать результат сложения в правую часть блока
        a_0[i] = G[i]

    # Объединить правую и левую части в один блок
    for i in range(4):
        out_data[i] = a_1[i]
        out_data[4 + i] = a_0[i]


def GOST_Magma_G_Fin(k, a, out_data):
    a_0 = [0] * 4  # Правая часть блока
    a_1 = [0] * 4  # Левая часть блока
    G = [0] * 4

    # Разделить 64-битный входной блок на две части
    for i in range(4):
        a_0[i] = a[4 + i]
        a_1[i] = a[i]

    # Применить преобразование g
    GOST_Magma_g(k, a_0, G)

    # Применить XOR результата преобразования g к левой части блока
    GOST_Magma_Add(a_1, G, G)

    # Скопировать результат сложения в левую часть блока
    for i in range(4):
        a_1[i] = G[i]
```

```
    # Объединить правую и левую части в один блок
    for i in range(4):
        out_data[i] = a_1[i]
        out_data[4 + i] = a_0[i]


def GOST_Magma_Encrypt(blk, out_blk):
    # Первое преобразование G
    GOST_Magma_G(iter_key[0], blk, out_blk)

    # Последующие (со второго по тридцать первое) преобразования G
    for i in range(1, 31):
        GOST_Magma_G(iter_key[i], out_blk, out_blk)

    # Последнее (тридцать второе) преобразование G
    GOST_Magma_G_Fin(iter_key[31], out_blk, out_blk)


def GOST_Magma_Decript(blk, out_blk):
    # Первое преобразование G с использованием
    # тридцать второго итерационного ключа
    GOST_Magma_G(iter_key[31], blk, out_blk)

    # Последующие (со второго по тридцать первое) преобразования G
    # (итерационные ключи идут в обратном порядке)
    for i in range(30, 0, -1):
        GOST_Magma_G(iter_key[i], out_blk, out_blk)

    # Последнее (тридцать второе) преобразование G
    # с использованием первого итерационного ключа
    GOST_Magma_G_Fin(iter_key[0], out_blk, out_blk)

iter_key = [bytearray(4) for _ in range(32)]  # Инициализация ключевого
расписания

def GOST_Magma_Expand_Key(key):
    # Формирование 8-ми 32-битных подключей
    for i in range(8):
        iter_key[i][:] = key[i * 4: (i + 1) * 4]


    #повторяем прошлый блок ещё 2 раза
    for j in range(2):
        for i in range(8):
            iter_key[8 * (j + 1) + i][:] = key[i * 4: (i + 1) * 4]

    for i in range(8):
        iter_key[-(i+1)][:] = iter_key[i][:]

# print("Transformation t")
```

```
def t(input_data):
    result = bytearray(4)
    GOST_Magma_T(input_data, result)
    return result

def add_xor(a,b):
    block_size=8
    c=[""]*block_size
    for i in range(block_size):
        c[i]=a[i]^b[i]
    return c



def g(k, a):
    result = bytearray(4)
    GOST_Magma_g(bytearray.fromhex(k), bytearray.fromhex(a), result)
    return result


key_example =
bytearray.fromhex("ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfe
ff")
GOST_Magma_Expand_Key(key_example)
# print("Key Schedule:", [''.join(format(x, '02x') for x in subkey) for subkey in
iter_key])

# A.2.4 Encryption Algorithm
def GOST_Magma_Encrypt_Example():
    plaintext = bytearray.fromhex("92def06b3c130a59")
    encrypted_block = bytearray(8)

    GOST_Magma_Encrypt(plaintext, encrypted_block)

    print("Open text:", ''.join(format(x, '02x') for x in plaintext))
    print("Encrypted Block:", ''.join(format(x, '02x') for x in encrypted_block))

GOST_Magma_Encrypt_Example()

def GOST_Magma_Decrypt_Example():
    # Шифртекст для расшифрования
    ciphertext = bytearray.fromhex("2b073f0494f372a0")
    decrypted_block = bytearray(8)

    GOST_Magma_Decript(ciphertext, decrypted_block)

    print("Ciphertext:", ''.join(format(x, '02x') for x in ciphertext))
    print("Decrypted Block:", ''.join(format(x, '02x') for x in decrypted_block))

# Пример расшифрования
GOST_Magma_Decrypt_Example()
```

Пример работы: