МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ЛАБОРАТОРНЫЕ РАБОТЫ ПО ПРЕДМЕТУ

«Программирование криптографических алгоритмов»

Выполнил:

Барышников С.С.
гр. 191-351

Преподаватель:

Бутакова Н.Г.

Москва 2021 г.

Содержание

Аннотация

Среда программирования: Visual Studio Code

Язык программирования: Python 3

Процедуры для запуска программы: $ python3 <имя_файла>.py

Пословица-тест: Время, приливы и отливы не ждут человека.

Текст для проверки работы: Вот пример статьи на тысячу символов. Это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазинах или для небольших информационных публикаций. В таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. Но можно и без него. На тысячу символов рекомендовано использовать один или два ключа и одну картину. Текст на тысячу символов это сколько примерно слов? Статистика показывает, что тысяча включает в себя сто пятьдесят или двести слов средней величины. Но, если злоупотреблять предлогами, союзами и другими частями речи на один или два символа, то количество слов неизменно возрастает. В копирайтерской деятельности принято считать тысячи с пробелами или без. Учет пробелов увеличивает объем текста примерно на сто или двести символов именно столько раз мы разделяем слова свободным пространством. Считать пробелы заказчики не любят, так как это пустое место. Однако некоторые фирмы и биржи видят справедливым ставить стоимость за тысячу символов с пробелами, считая последние важным элементом качественного восприятия. Согласитесь, читать слитный текст без единого пропуска, никто не будет. Но большинству нужна цена за тысячу знаков без пробелов.

Интерфейс: #в разработке#

Постоянный модуль

Код модуля base.py используемый для предотвращения дублирования кода, используется во всех последующих программах:

```
import re

alphabet = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"

dict = {'.': 'тчк', ',': 'зпт'}


def replace_all_to(input_text, dict):
    input_text = input_text.replace(' ', '')
    for i, j in dict.items():
        input_text = input_text.replace(i, j)
    return input_text


def replace_all_from(input_text, dict):
    for i, j in dict.items():
        input_text = input_text.replace(j, i)
    return input_text


def file_to_string(name):
    with open(name) as f:
        input_short_text = " ".join([l.rstrip() for l in f]) + ' '
    return input_short_text.lower()


def input_for_cipher_short():
    return replace_all_to(file_to_string('short.txt'), dict)


def input_for_cipher_long():
    return replace_all_to(file_to_string('long.txt'), dict)


def output_from_decrypted(decrypted_text):
    return replace_all_from(decrypted_text, dict)
```

Е: ШИФРЫ ГАММИРОВАНИЯ

13.Одноразовый блокнот К.Шеннона

Популярность поточных шифров можно связывать с работой Клода Шеннона, посвященной анализу одноразовых гамма-блокнотов. Название «одноразовый блокнот» стало общепринятым в годы Второй мировой войны, когда для шифрования широко использовались бумажные блокноты.

Одноразовый блокнот использует длинную шифрующую последовательность, которая состоит из случайно выбираемых бит или наборов бит (символов). Шифрующая последовательность побитно или посимвольно накладывается на открытый текст, имеет ту же самую длину, что и открытое сообщение, и может использоваться только один раз (о чем свидетельствует само название шифрсистемы); ясно, что при таком способе шифрования требуется огромное количество шифрующей гаммы.

Открытый текст сообщения ш записывают как последовательность бит или символов m = momi...mn_i, а двоичную или символьную шифрующую последовательность к той же самой длины - как k = koki...k,,_|.

Шифртекст c = c0cl...cn.i определяется соотношением Cj = mi Шк, при 0

Код программы:

```
import random
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted

alphabet = alphabet.replace(' ', '')
alphabet_lower = {}
i = 0
while i < (len(alphabet)):
    alphabet_lower.update({alphabet[i]: i})
    i += 1
def get_key(d, value):
    for k, v in d.items():
        if v == value:
            return k


def shenon_encode(msg):
    msg_list = list(msg)
    msg_list_len = len(msg_list)
    msg_code_bin_list = list()
    for i in range(len(msg_list)):
        msg_code_bin_list.append(alphabet_lower.get(msg_list[i]))

    key_list = list()
```

```
    for i in range(msg_list_len):
        key_list.append(random.randint(0, 32))

    cipher_list = list()
    for i in range(msg_list_len):
        m = int(msg_code_bin_list[i])
        k = int(key_list[i])
        cipher_list.append(int(bin(m ^ k), base=2))
    return cipher_list, key_list


def shenon_decode(msg, key_list):
    decipher_list = list()
    msg_list_len = len(msg)
    for i in range(msg_list_len):
        c = int(msg[i])
        k = int(key_list[i])
        decipher_list.append(int(bin(c ^ k), base=2))
    deciphered_str = ""
    for i in range(len(decipher_list)):
        deciphered_str += get_key(alphabet_lower, decipher_list[i])
    return deciphered_str


short_encoded = shenon_encode(input_for_cipher_short())
short_decoded = shenon_decode(short_encoded[0], short_encoded[1])

long_encoded = shenon_encode(input_for_cipher_long())
long_decoded = shenon_decode(long_encoded[0], long_encoded[1])

print(f'''
Одноразовый блокнот:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{short_encoded[0]}
Ключ:
{short_encoded[1]}

Расшифрованный текст:
{output_from_decrypted(short_decoded)}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{long_encoded[0]}
Ключ:
{long_encoded[1]}

Расшифрованный текст:
{output_from_decrypted(long_decoded)}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab05_13_shenon.py

Одноразовый блокнот:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
[20, 24, 3, 1, 54, 21, 5, 27, 10, 6, 19, 22, 12, 23, 30, 22, 12, 20, 31, 13,
30, 6, 17, 14, 21, 0, 0, 4, 20, 10, 7, 10, 0, 3, 15, 9, 26, 31, 26]
Ключ:
[22, 9, 6, 12, 22, 29, 21, 8, 26, 23, 26, 26, 5, 21, 2, 31, 3, 7, 19, 4, 28,
26, 31, 11, 18, 4, 20, 23, 12, 15, 11, 5, 2, 6, 4, 9, 9, 7, 17]

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
[12, 47, 30, 29, 1, 28, 12, 13, 17, 28, 14, 8, 8, 13, 23, 25, 7, 14, 25, 7,
49, 56, 27, 11, 13, 16, 10, 24, 9, 14, 3, 26, 22, 6, 9, 6, 7, 0, 6, 5, 7, 31,
7, 12, 12, 11, 1, 15, 32, 25, 24, 4, 31, 17, 7, 9, 2, 8, 23, 9, 17, 17, 48,
4, 3, 19, 1, 10, 1, 4, 27, 9, 25, 23, 15, 17, 29, 12, 13, 3, 47, 21, 22, 12,
19, 19, 58, 10, 8, 12, 11, 14, 31, 31, 31, 9, 6, 18, 1, 49, 3, 10, 27, 3, 19,
21, 26, 16, 13, 2, 5, 21, 15, 26, 5, 5, 5, 32, 29, 28, 9, 14, 25, 3, 22, 15,
6, 14, 34, 14, 7, 15, 18, 0, 29, 1, 27, 14, 12, 46, 16, 27, 2, 4, 27, 16, 16,
10, 12, 0, 21, 1, 28, 24, 17, 1, 8, 27, 19, 13, 11, 16, 20, 9, 1, 34, 23, 19,
25, 7, 20, 11, 1, 14, 23, 9, 10, 21, 4, 10, 29, 6, 3, 6, 6, 9, 23, 0, 4, 0,
14, 14, 9, 12, 15, 1, 20, 14, 27, 5, 20, 0, 13, 13, 25, 27, 1, 7, 7, 2, 15,
6, 7, 27, 10, 23, 2, 9, 31, 21, 22, 25, 22, 23, 36, 23, 31, 10, 18, 8, 19,
28, 21, 8, 4, 23, 1, 22, 29, 31, 28, 27, 31, 22, 21, 3, 23, 30, 7, 11, 0, 18,
31, 20, 4, 20, 25, 27, 19, 8, 45, 26, 9, 14, 7, 16, 20, 15, 23, 26, 11, 16,
29, 23, 7, 24, 8, 4, 4, 9, 9, 14, 22, 19, 15, 23, 3, 2, 15, 9, 19, 1, 9, 11,
9, 17, 14, 27, 12, 25, 7, 21, 2, 8, 31, 13, 23, 44, 13, 5, 9, 29, 23, 20, 11,
25, 2, 5, 2, 4, 26, 23, 20, 27, 9, 27, 30, 24, 14, 7, 17, 23, 4, 0, 13, 17,
33, 22, 10, 20, 9, 13, 24, 16, 27, 1, 3, 11, 21, 9, 17, 20, 17, 20, 19, 26,
21, 24, 1, 17, 0, 9, 22, 9, 19, 4, 44, 22, 25, 28, 26, 28, 5, 29, 12, 3, 12,
8, 22, 18, 5, 7, 14, 8, 27, 16, 12, 15, 29, 30, 6, 20, 17, 29, 15, 14, 18,
24, 7, 22, 28, 38, 15, 28, 6, 16, 4, 6, 21, 23, 9, 22, 9, 1, 18, 26, 52, 22,
19, 21, 25, 37, 15, 28, 31, 8, 30, 36, 19, 17, 14, 12, 26, 5, 4, 30, 6, 0, 5,
25, 0, 14, 25, 27, 1, 36, 14, 15, 6, 11, 31, 13, 17, 1, 31, 27, 28, 7, 12,
22, 8, 28, 17, 22, 16, 25, 25, 28, 21, 29, 13, 0, 20, 17, 13, 23, 23, 37, 15,
8, 58, 4, 8, 9, 27, 7, 5, 19, 21, 13, 21, 25, 24, 19, 4, 2, 26, 25, 12, 11,
24, 26, 8, 19, 11, 11, 31, 35, 1, 9, 26, 22, 2, 25, 26, 60, 24, 2, 49, 15,
22, 27, 14, 3, 29, 12, 7, 13, 7, 31, 2, 12, 22, 13, 14, 17, 5, 6, 31, 21, 21,
14, 14, 25, 4, 7, 30, 25, 27, 10, 8, 16, 7, 14, 6, 29, 29, 17, 2, 31, 15, 12,
14, 19, 17, 31, 21, 25, 16, 10, 26, 19, 9, 31, 22, 15, 32, 18, 1, 27, 3, 0,
17, 5, 47, 10, 11, 16, 4, 29, 4, 4, 17, 21, 10, 14, 10, 10, 13, 50, 15, 15,
2, 19, 15, 10, 18, 7, 29, 48, 13, 15, 13, 42, 7, 1, 13, 7, 0, 27, 4, 0, 16,
13, 17, 4, 34, 1, 12, 7, 6, 8, 19, 13, 10, 9, 22, 25, 10, 23, 12, 23, 30, 29,
40, 10, 16, 2, 16, 2, 7, 13, 31, 27, 15, 16, 11, 0, 19, 4, 10, 31, 26, 16,
```

```
24, 56, 2, 18, 8, 0, 30, 24, 5, 11, 0, 6, 31, 19, 10, 31, 8, 23, 5, 24, 26,
6, 30, 11, 13, 25, 16, 17, 13, 6, 30, 21, 15, 7, 1, 9, 2, 15, 12, 14, 0, 10,
6, 31, 5, 29, 26, 5, 30, 12, 28, 4, 18, 8, 1, 13, 24, 24, 7, 8, 21, 27, 25,
17, 18, 2, 27, 16, 3, 4, 12, 22, 45, 15, 0, 10, 14, 29, 34, 31, 12, 22, 31,
21, 26, 2, 16, 16, 11, 15, 0, 6, 23, 26, 25, 2, 2, 28, 16, 3, 24, 2, 25, 17,
26, 10, 26, 16, 21, 25, 27, 4, 18, 21, 23, 6, 0, 18, 30, 6, 24, 17, 1, 16,
25, 22, 20, 15, 30, 37, 29, 4, 21, 54, 6, 18, 3, 8, 6, 12, 23, 27, 15, 27, 4,
51, 29, 30, 7, 14, 31, 5, 0, 18, 18, 29, 31, 5, 27, 19, 21, 30, 4, 4, 21, 31,
31, 46, 3, 16, 18, 4, 12, 25, 5, 22, 28, 7, 11, 9, 29, 23, 29, 22, 11, 12,
10, 7, 24, 29, 35, 7, 13, 10, 12, 12, 17, 17, 1, 14, 17, 22, 19, 21, 20, 16,
0, 20, 17, 51, 13, 30, 10, 24, 2, 25, 20, 21, 22, 17, 13, 23, 13, 25, 29, 41,
25, 20, 26, 23, 25, 12, 20, 1, 26, 34, 11, 16, 20, 29, 20, 17, 44, 5, 21, 13,
27, 7, 17, 50, 21, 0, 13, 31, 53, 31, 27, 21, 19, 26, 24, 17, 41, 5, 30, 21,
8, 14, 16, 45, 3, 22, 37, 17, 21, 20, 25, 12, 45, 26, 14, 14, 1, 50, 18, 21,
22, 5, 18, 24, 17, 24, 13, 28, 21, 10, 16, 6, 52, 1, 30, 35, 17, 23, 2, 25,
1, 20, 19, 9, 20, 7, 18, 26, 3, 9, 11, 31, 11, 22, 26, 7, 11, 15, 16, 31, 20,
8, 11, 20, 11, 5, 0, 27, 24, 1, 9, 30, 22, 23, 14, 7, 41, 28, 31, 16, 30, 21,
10, 28, 19, 10, 14, 25, 10, 2, 19, 21, 27, 0, 11, 26, 26, 27, 30, 12, 21, 4,
10, 0, 10, 25, 13, 11, 20, 13, 31, 19, 27, 15, 29, 22, 8, 21, 31, 21, 11, 9,
0, 15, 10, 7, 4, 17, 14, 16, 24, 9, 22, 3, 55, 17, 3, 6, 46, 23, 21, 14, 18,
18, 1, 4, 18, 25, 25, 0, 16, 7, 8, 19, 51, 25, 30]
Ключ:
[14, 32, 13, 13, 16, 21, 1, 8, 0, 14, 29, 8, 27, 16, 30, 23, 7, 29, 5, 21,
17, 32, 15, 25, 4, 29, 8, 23, 5, 1, 1, 9, 14, 13, 23, 21, 8, 4, 9, 23, 20,
31, 20, 3, 20, 5, 14, 2, 32, 21, 29, 10, 2, 26, 14, 3, 17, 13, 28, 27, 2, 25,
32, 23, 12, 3, 18, 3, 12, 4, 23, 20, 23, 24, 31, 30, 25, 26, 2, 7, 15, 15,
31, 6, 23, 31, 26, 1, 8, 29, 24, 1, 7, 26, 20, 26, 9, 16, 1, 32, 12, 8, 25,
10, 29, 6, 31, 1, 3, 7, 22, 28, 3, 19, 8, 5, 6, 32, 21, 21, 7, 14, 15, 10,
26, 6, 2, 2, 2, 0, 2, 14, 29, 12, 0, 24, 18, 24, 5, 32, 5, 20, 19, 9, 27, 7,
25, 5, 2, 14, 9, 23, 12, 12, 16, 13, 1, 16, 19, 26, 2, 26, 7, 17, 10, 32, 4,
19, 18, 8, 25, 24, 4, 5, 5, 26, 15, 4, 1, 14, 22, 9, 2, 26, 4, 9, 18, 19, 5,
15, 2, 11, 12, 8, 13, 21, 2, 7, 23, 12, 7, 17, 11, 27, 25, 26, 9, 7, 16, 7,
13, 15, 8, 26, 22, 15, 12, 6, 16, 17, 31, 23, 6, 24, 32, 31, 31, 9, 29, 4,
28, 30, 26, 3, 23, 15, 10, 24, 18, 18, 19, 28, 17, 25, 28, 2, 18, 22, 9, 14,
3, 29, 12, 12, 15, 26, 25, 8, 15, 26, 13, 2, 29, 28, 14, 29, 22, 0, 27, 21,
9, 1, 24, 28, 8, 21, 13, 10, 0, 6, 11, 14, 24, 28, 6, 5, 19, 13, 3, 20, 27,
14, 11, 11, 26, 12, 1, 31, 5, 23, 14, 25, 11, 12, 29, 13, 28, 32, 18, 29, 9,
20, 24, 16, 5, 13, 9, 5, 19, 23, 19, 25, 0, 8, 17, 16, 13, 29, 5, 21, 2, 25,
4, 19, 17, 3, 1, 14, 30, 6, 0, 0, 26, 31, 23, 14, 1, 21, 6, 6, 3, 31, 30, 24,
14, 17, 26, 8, 16, 24, 13, 12, 7, 7, 28, 22, 32, 25, 27, 15, 2, 23, 23, 14,
12, 16, 5, 26, 5, 27, 14, 7, 30, 7, 16, 16, 4, 19, 31, 30, 3, 7, 25, 13, 28,
22, 1, 23, 20, 10, 14, 6, 23, 28, 4, 27, 8, 25, 13, 23, 12, 5, 11, 19, 23,
27, 20, 4, 0, 26, 9, 5, 28, 1, 27, 13, 12, 4, 0, 24, 2, 5, 30, 7, 1, 12, 21,
9, 23, 21, 15, 12, 11, 10, 4, 32, 0, 10, 12, 9, 26, 1, 24, 25, 22, 21, 0, 20,
20, 29, 6, 19, 25, 6, 3, 28, 11, 16, 28, 21, 1, 15, 0, 1, 2, 4, 6, 32, 14, 4,
26, 23, 21, 25, 10, 2, 1, 31, 26, 14, 21, 20, 17, 27, 20, 17, 8, 22, 19, 3,
24, 23, 1, 26, 15, 26, 11, 32, 8, 4, 19, 14, 2, 11, 9, 28, 21, 11, 32, 10,
14, 18, 0, 3, 18, 8, 14, 3, 14, 19, 11, 8, 20, 13, 28, 24, 8, 4, 16, 25, 21,
6, 30, 10, 23, 8, 21, 22, 23, 3, 16, 21, 21, 29, 4, 18, 15, 29, 13, 29, 1, 9,
7, 27, 28, 26, 27, 23, 31, 8, 21, 27, 24, 31, 4, 28, 32, 23, 18, 8, 27, 11,
```

```
19, 14, 32, 26, 2, 1, 4, 23, 23, 1, 0, 7, 1, 1, 0, 14, 8, 18, 28, 10, 14, 14,
1, 5, 0, 20, 20, 32, 28, 6, 3, 10, 20, 14, 31, 31, 9, 8, 4, 19, 13, 30, 13,
22, 2, 25, 5, 21, 22, 25, 28, 12, 15, 5, 22, 20, 3, 30, 0, 30, 31, 24, 32,
25, 8, 9, 4, 26, 2, 30, 15, 10, 0, 17, 14, 12, 28, 6, 30, 29, 31, 28, 17, 32,
11, 16, 8, 5, 13, 23, 4, 16, 5, 11, 12, 22, 1, 13, 27, 23, 21, 9, 19, 11, 27,
26, 3, 22, 30, 17, 31, 21, 17, 28, 3, 14, 5, 11, 7, 29, 31, 7, 18, 3, 11, 29,
10, 17, 21, 7, 23, 1, 25, 10, 28, 7, 19, 30, 23, 20, 26, 3, 26, 10, 25, 25,
31, 30, 10, 16, 11, 0, 9, 26, 13, 10, 13, 24, 2, 18, 32, 31, 30, 20, 16, 20,
21, 6, 30, 12, 6, 31, 17, 9, 5, 9, 8, 2, 12, 14, 3, 1, 23, 15, 10, 9, 17, 24,
2, 25, 6, 25, 8, 25, 2, 4, 24, 7, 5, 30, 2, 14, 24, 26, 1, 24, 1, 31, 31, 6,
16, 32, 17, 27, 20, 22, 21, 26, 19, 27, 21, 12, 28, 16, 15, 16, 26, 32, 18,
14, 19, 28, 12, 10, 5, 31, 23, 15, 12, 10, 8, 11, 30, 17, 0, 10, 21, 20, 16,
32, 6, 27, 29, 23, 3, 8, 25, 19, 9, 14, 26, 4, 1, 30, 28, 31, 26, 11, 3, 5,
17, 25, 3, 20, 31, 26, 29, 12, 19, 20, 5, 2, 24, 20, 15, 24, 6, 3, 0, 22, 24,
32, 16, 12, 25, 23, 11, 20, 27, 7, 5, 12, 5, 23, 30, 5, 15, 9, 1, 0, 8, 30,
20, 14, 27, 13, 21, 32, 25, 0, 5, 18, 21, 20, 32, 5, 24, 4, 19, 23, 2, 32,
13, 9, 30, 31, 21, 15, 20, 7, 31, 31, 28, 31, 32, 0, 28, 21, 15, 0, 12, 32,
29, 26, 32, 28, 16, 26, 10, 3, 32, 17, 14, 22, 4, 32, 1, 23, 19, 11, 28, 23,
18, 23, 15, 19, 7, 26, 1, 15, 20, 18, 23, 3, 2, 15, 9, 11, 14, 23, 31, 9, 6,
14, 1, 31, 17, 20, 3, 15, 24, 14, 19, 20, 11, 28, 13, 13, 24, 1, 24, 26, 23,
15, 19, 30, 19, 19, 26, 31, 19, 31, 11, 3, 32, 18, 16, 19, 17, 5, 27, 19, 3,
30, 28, 18, 10, 10, 3, 6, 21, 9, 0, 9, 21, 21, 27, 13, 1, 0, 15, 19, 25, 1,
6, 5, 27, 12, 16, 31, 6, 22, 20, 24, 26, 6, 29, 1, 5, 29, 7, 1, 10, 16, 1,
31, 14, 24, 24, 26, 10, 17, 23, 9, 23, 14, 32, 23, 30, 1, 16, 19, 4, 12, 2,
8, 22, 1, 21, 11, 7, 17, 32, 1, 21]

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходя
щийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпублик
аций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.но
можноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднука
ртину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотыс
ячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблят
ьпредлогами,союзамииидругимичастямиречинаодинилидвасимвола,токоличествословнеи
зменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиили
без.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностоо
лькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,
таккакэтопустоеместо.однаконекоторыефирмыибиржividятсправедливымставитьстоимо
стьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговоспр
иятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольши
нствунужнаценазатысячузнаковбезпробелов.
```

14.Гаммирование ГОСТ 28147-89

При работе ГОСТ 28147-89 в режиме гаммирования описанным выше образом формируется криптографическая гамма, которая затем побитно складывается по модулю 2 с исходным открытым текстом для получения шифротекста. Шифрование в режиме гаммирования лишено недостатков, присущих режиму простой замены. Так, даже идентичные блоки исходного текста дают разный шифротекст, а для текстов с длиной, не кратной 64 бит, "лишние" биты гаммы отбрасываются. Кроме того, гамма может быть выработана заранее, что соответствует работе шифра в поточном режиме.

Код программы:

```
# -*- coding:utf-8 -*-
import sys
import numpy.random
import itertools
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted
import binascii


class GostCrypt(object):
    def __init__(self, key, sbox):
        self._key = None
        self._subkeys = None
        self.key = key
        self.sbox = sbox

    @staticmethod
    def _bit_length(value):
        return len(bin(value)[2:])

    @property
    def key(self):
        return self._key

    @key.setter
    def key(self, key):
        self._key = key
        self._subkeys = [(key >> (32 * i)) & 0xFFFFFFFF for i in range(8)]


    def _f(self, part, key):
        temp = part ^ key
        output = 0
        for i in range(8):
            output |= ((self.sbox[i][(temp >> (4 * i)) & 0b1111]) << (4 * i))
        return ((output >> 11) | (output << (32 - 11))) & 0xFFFFFFFF
```

```
    def _decrypt_round(self, left_part, right_part, round_key):
        return left_part, right_part ^ self._f(left_part, round_key)

    def encrypt(self, plain_msg):
        def _encrypt_round(left_part, right_part, round_key):
            return right_part, left_part ^ self._f(right_part, round_key)

        left_part = plain_msg >> 32
        right_part = plain_msg & 0xFFFFFFFF
        for i in range(24):
            left_part, right_part = _encrypt_round(left_part, right_part, sel
f._subkeys[i % 8])
        for i in range(8):
            left_part, right_part = _encrypt_round(left_part, right_part, sel
f._subkeys[7 - i])
        return (left_part << 32) | right_part

    def decrypt(self, crypted_msg):
        def _decrypt_round(left_part, right_part, round_key):
            return right_part ^ self._f(left_part, round_key), left_part

        left_part = crypted_msg >> 32
        right_part = crypted_msg & 0xFFFFFFFF
        for i in range(8):
            left_part, right_part = _decrypt_round(left_part, right_part, sel
f._subkeys[i])
        for i in range(24):
            left_part, right_part = _decrypt_round(left_part, right_part, sel
f._subkeys[(7 - i) % 8])
        return (left_part << 32) | right_part


sbox = [numpy.random.permutation(l) for l in itertools.repeat(list(range(16))
, 8)]
sbox = (
    (4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3),
    (14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9),
    (5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11),
    (7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3),
    (6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2),
    (4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14),
    (13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12),
    (1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12),
    )

key = 183182793879123879127893789123798218793879782387932788723783298329823982
023031
```

```
text_short = input_for_cipher_short().encode().hex()
text_short = int(text_short, 16)

gost_short = GostCrypt(key, sbox)

encode_text_short = gost_short.encrypt(text_short)
decode_text_short = gost_short.decrypt(encode_text_short)
decode_text_short = bytes.fromhex(hex(decode_text_short)[2::]).decode('utf-
8')


text_long = input_for_cipher_long().encode().hex()
text_long = int(text_long, 16)

gost_long = GostCrypt(key, sbox)

encode_text_long = gost_long.encrypt(text_long)
decode_text_long = gost_long.decrypt(encode_text_long)
decode_text_long = bytes.fromhex(hex(decode_text_long)[2::]).decode('utf-8')


print(f'''
Гост 28147-89:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{encode_text_short}

Расшифрованный текст:
{output_from_decrypted(decode_text_short)}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{encode_text_long}

Расшифрованный текст:
{output_from_decrypted(decode_text_long)}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab05_14_gost89.py

Гост 28147-89:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
567540261451836962860566905831140964633059962168239725400849570714503615166865346335420862814981341444458694338340017794062562381816998355678753967115290745875600125177595186371409630906 82

Расшифрованный текст:
время,приливыиотливынеждутчеловека.
```

```
ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
266050417938476356087017230014873643206160447996114515129091774959747033405943890343897142869704450815393683605023684873888004909829680452295028718546978021649941177826880151732274533643864532320796630272450746837347468428306368235089557119542257601405589120123495191711757441077720644314230550270347750883998451898671458314340016425755013995595170529190526024934975357719363564132719076103190529035599529700114243798479936214759685480345957150176507752611049845995003758144411130130178759699170420308485113983727745509962073921913391810397788675115132457776580708464051846373071577643730197234727027073854195630603068210347317635511545038247142648123133161307062640743932444830284935751942801836958416328466946774416647040159871667643738658517814221245915214928246803162017812436951725385369568312910693127721305386590770266282001789890178271024558112846441974461477282273708237956958284488468419049021450965726813580738701067634703398359232464360260708890382002875053592506820582408502620331704501617095262512488892027835394566041168480590398364827945085706355241175058417003907247019752582482794061075722227675822081892404516659761523826038518227529293356459089601327841046712363123251360022267584062361921103062944008184059512152151700998698904096770829401135020739563435815427903679578218875067810914095989673219364329166908308361570998442869287890027766834584419183407652747042511617269389753273012998279245334616877159640347816015296464295681617647172047594328251693989022067560411048663904138961949717524467797892198348685209617903445304383592590876810894190958189473679242424610526622461775514367598680609644131641613792985187497438227966239919276204302869135347925244725013856160622834337882661360341824413825968528286788154642488982426559565499876930164748573179670457658054023039292389178257684534213982610204775980600094649100927108067273892323755053371019809720521867205334093895534822121423383888251650643168732003624170059793831256089491353050782028804944025904299184235348674984813250027251862086720688049102194441287254123131728573159600922255684796434774080500238892291475276783954053000145147885599279208235047400922773599948384461446382922212048102735317321789072559748723739437771744892787203600302455132063962866809231326655919329574793816888817133914475501975124214140801556180788374367162542394957389493526043951151164131904106002370372557788499423181314004878679937166122458155086667883866115359834887008407878066080617386495957544097625981039926098512363901470340589971200819260151401068029365898208739199000231395968707522505328350595047697302991894042689424537180277739107303643644783097835573010019102998769286433822677797732973441605094074880504351040539686018039879610535083300978658667265664391967425571351647822609084567301283329117759836853207234726828064964608886180317212877329147310950651510665931483604003665609249042923766617819843259861537060491859686395270315553393280970861384935830664056615086318518792532621575498933363420858503794211694857078464123907640827380597754238631523652222073583704208486423290328961434777281474628730883012418585005183526704604591722947806264958533139152742817759695979393904085889210131910527536391330455357053721771502167641015044650209047454483953632063217780184666438748654653608742685666423054367766118254669621964271957227771110446560009008260085554665704132125404748670093454836785013948460818856547409223615702492801811515171154353076248616600260838281861827510318136996678149054655293501452211334327394481409806145693576184764835508628555563178166027833341769254061164490913955537268308991417425837357068599913797622693757837464689346947323941
```

```
56558064456440375957265656396640816066359824776600968566072302055241997176140
08226865423733917323140685928071900336012766976449108365776366982263306482365
53701481899448499076055004435636081490697301414470015727863530449650719384883
20817393982327350914870757149168519557298244867969723184161812866514205609360
54534723880151398796895076146362679321996134948186011756944951068239657356798
52248453340323322884737319847422163663683329727707507865218141639579913178721
46879701156367505884840443242033743649284403534308730028565578448899028468749
36816595913309923492206525652964107796613497401751530613052886454175587861865
33328282877556855948224417771287356978869466410912431116815452381165518582443
06426392836846104064731076842543578816818530479152156759120290446042559868977
98394652898650750455488036258790990498804577566340962162875175833798508271140
94281279725186959719047716798991463990201696001595394956661030446430788409672
45774105859770442044412377244765493396419787742155086335373263474448018184154
88987374615175848442941209032049881504640534180612881439490586821993445040088
61383378392402642556140683240706965932049456398214008601555594844595489325508
40102882387267858402968600171117615737683143477720170728529983369912503772826 9
83172074521408101157630622727122622993941598645103078049746802620829309937928
82105434472845097445767818724620636250333319570116562701500864340123631182252
02497856559102049032543407053333746197469855990322303022144902536648886278160
40235778291362135957003710956166661912952801584275480948432041064323813779756
93276853993148974688913227746868482199358865760126313652003609854882178324694

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходя
щийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпублик
аций.втакомтекстередкобываетболееедвухилитрёхабзацевиобычноодинподзаголовок.но
можноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднука
ртину.текстнатысячусимволовэтоскольkопримернослов.статистикапоказывает,чтотыс
ячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблят
ьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеи
зменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиили
без.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименносто
лькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,
таккакэтопустоеместо.однаконекоторыефирмыибиржевидятсправедливымставитьстоимо
стьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосп
риятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольши
нствунужнаценазатысячузнаковбезпробелов.
```